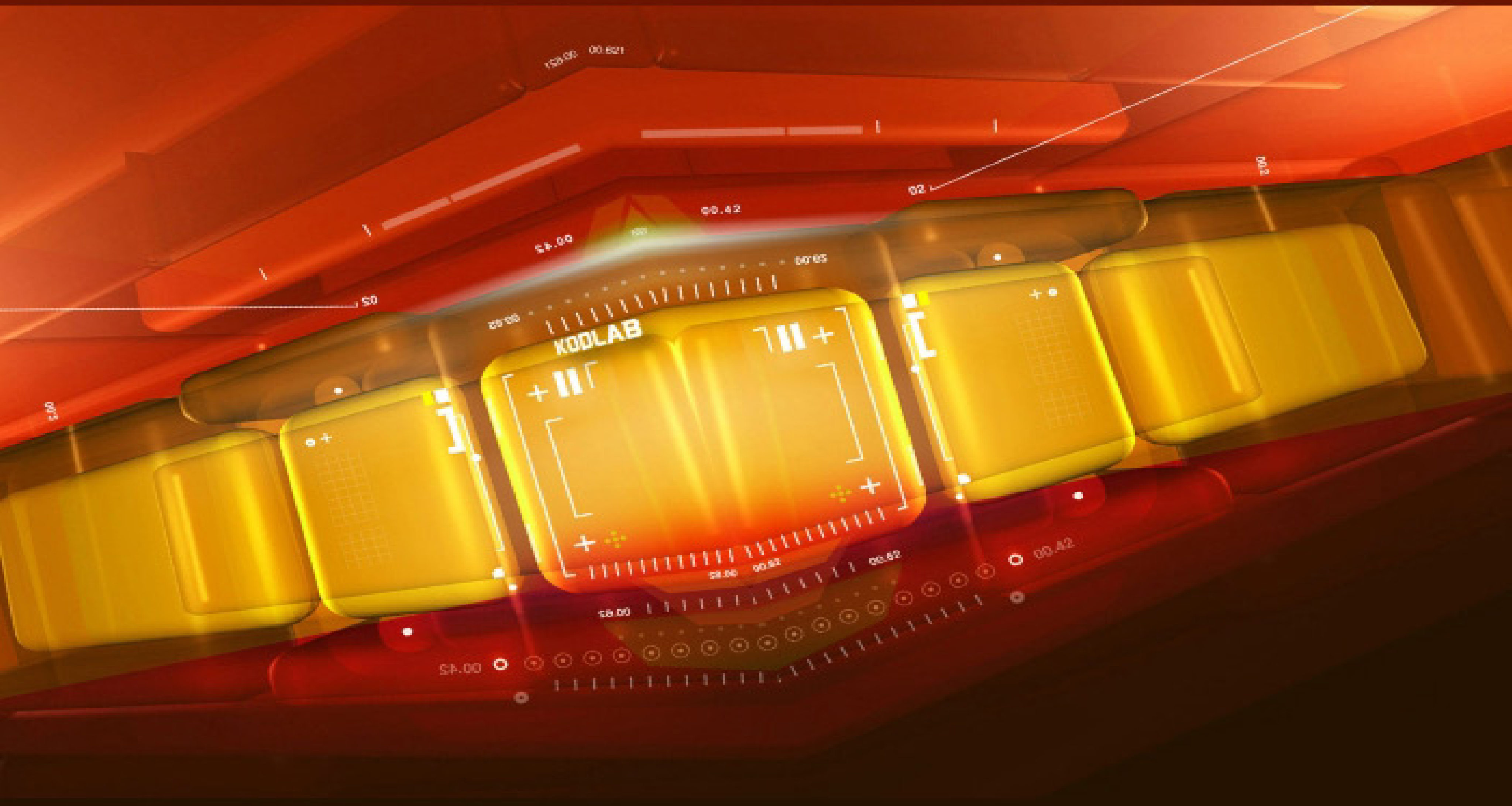
İbrahim ÇELİKBİLEK



HER YÖNÜYLE HTML5

HTML5 • JavaScript • CSS3

KODLAB®
www.kodlab.com

İÇİNDEKİLER

HER YÖNÜYLE HTML5

İBRAHİM
ÇELİKBİLEK

KODLAB®
Yayın Dağıtım Yazılım ve Eğitim
Hizmetleri San. ve Tic. Ltd. Şti.

KODLAB® 53

HER YÖNÜYLE HTML5

İBRAHİM ÇELİKBİLEK

ISBN 978-605-4205-48-6
Yayıncılık Sertifika No: 13206

1. Baskı: Haziran 2011

Yayın Yönetmeni: Uğur Gelişken
Sayfa Düzeni: Mehmet Öztürk
Satış: Hüseyin Üstünel
Baskı: Şefik Basım ve Yayıncılık San. Tic. Ltd. Şti. Tel: (212) 549 62 62

KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti.
Alemdar Mah. Salkım Söğüt Sok. Ağa İş Hanı
No: 20 Kat:1 D: 3 Sultanahmet / İSTANBUL

tel: (212) 514 55 66 | **fax:** (212) 514 66 61
e-posta: bilgi@kodlab.com | **web:** www.kodlab.com

İBRAHİM ÇELİKBİLEK

1980 Kars doğumludur. 2002 yılında Fırat Üniversitesi T.E.F'ni bitirdi. Çeşitli eğitim kuruluşlarında yazılım uzmanlığı ile ilgili dersler verdi. 2005 yılında Suşehri Atatürk Endüstri Meslek Lisesine Bilgisayar Öğretmeni olarak atandı. 2009 Yılından itibaren bu görevini Kocaeli'de sürdürmektedir. JavaScript, CSS, DHTML, AJAX, ASP.NET ve C# konularında seminerler verdi ve makaleler yazdı. Web programlama alanında birçok uygulama geliştirdi. Ayrıca yazarımızın daha önce KODLAB'tan JavaScript ve XHTML ve CSS kitapları çıkmıştır.

Yazarımız hakkında daha detaylı bilgi almak için; *http://www.ibrahimcelikbilek.com* adresini ziyaret edebilirsiniz.

Önsöz

Kitabın siz okurlarım için faydalı olması dileğiyle sözlerime başlamak istiyorum.

Kitap içerisinde HTML5 yapısı ile birlikte bu yapı içerisinde bulunan yeni elemanlar, özellikler ve HTML5'in duyurulması ile beraber önemi artan ek teknolojiler hakkında geniş bilgiler bulabilirsiniz.

Konuları iyice kavrayabilmek için; kitap içinde verilen örnekleri dikkatlice inceleyip uygulamalı ve bu örnekleri referans alarak yeni uygulamalar geliştirmeye çalışmalısınız.

HTML5; html dili için yeni stratejiler ve hedefler ortaya koyan ve farklı web tarayıcıları için standardizasyonu amaçlayan yeni nesil bir sürüm olarak karşımıza çıkmaktadır. Olaya bu açıdan yaklaşırsak; bu dilin önümüzdeki süreçte hızlı bir şekilde gelişeceğini ve popülaritesini daha da arttıracağını görebiliriz. Aslında dilin sadece kendisi değil kullandığı ek teknolojilerde gelişime açık olmakla beraber bu dile büyük güç katmaktadır.

Kitap içerisindeki konuların daha iyi anlaşılması bakımından az da olsa JavaScript ve CSS bilgilerine sahip olmanız gerekmektedir. Çünkü HTML5 yapısının kullandığı programlama dili JavaScript'tir. Bu açıdan HTML5 ile beraber JavaScript dilinin hak ettiği konuma geldiğini düşünmekteyim.

HTML5 ya da diğer programlama konuları ile ilgili sorularınızı www.kodlab.com sitesinde bulunan yazara sor kısmından ya da *www.ibrahimcelikbilek.com* adresinden bana iletebilirsiniz.

İbrahim Çelikbilek

Gerekli Programlar

Kitabın siz okurlarım için faydalı olması dileğiyle sözlerime başlamak istiyorum.

Kitap içerisindeki konuları anlatılırken temelde kod mantığını kavramanız hedeflenmiştir. Kodları öğrendikten sonra uygulamaları istediğiniz herhangi bir programda, hatta bir metin editöründe bile geliştirebilirsiniz. Profesyonel uygulamalar oluştururken; gerek tasarım ekranı, gerek sunduğu yardımcı araçlar ve diğer özellikleri ile gelişmiş web sayfası editörlerini kullanmak işinizi kolaylaştıracaktır.

Kitabı verimli bir şekilde takip etmek için kullanabileceğiniz programlar şunlardır:

- **Aptana Studio 3:** *www.aptana.com* adresinden indirebilirsiniz.
- **Expression Web 4:** Deneme sürümünü *www.microsoft.com/expression/* adresinden indirebilirsiniz.
- **Dreamweaver CS5.5:** Deneme sürümünü www.adobe.com adresinden indirebilirsiniz.

Merhum Babam Ali Osman ÇELİKBİLEK'in anısına...

HTML5'E GİRİŞ

1

Genel Bakış

HTML5, html dilinin en son sürümüdür. Modern web uygulamaları geliştirmek için yeni eklentiler ve özellikler sunar. HTML5; html dili için yeni stratejiler ve hedefler ortaya koyan ve farklı web tarayıcıları için standardizasyonu amaçlayan yeni nesil bir sürüm olarak karşımıza çıkmaktadır. HTML5 ile html dilinin işlevselliği, programlama ve sunum gücü artmıştır. HTML5'e yeni eklenen yapısal elemanlar sayesinde fazla kod yazmadan sitenizin görsel tasarımını yapabilir, JavaScript ve CSS3 teknolojilerini daha etkili bir şekilde kullanabilirsiniz. HTML5 tasarımcıların web sayfalarında zengin medya içerikleri kullanmalarına ve interaktif web uygulamaları geliştirebilmelerine olanak sağlayan yeni elemanlar, özellikler ve teknolojiler barındırır.

HTML5 içinde bulunan tüm teknolojiler aslında HTML5 ile beraber yeni oluşturulan ve duyurulan teknolojiler değildir. W3C tarafından önceden oluşturulmuş ve modern web tarayıcıları tarafından kısmen desteklenen bir takım teknolojilerde (Örneğin; *SVG-Scalable Vector Graphics*) HTML5 içerisine dahil edilmiştir. W3C HTML5 dilini geliştirmeye devam etmektedir bu açıdan ilerleyen zaman dilimlerinde dile yeni eklentilerin olması muhtemeldir. Ayrıca HTML5 dilinin diğer bir avantajı da, HTML5 dili içinde bulunmayan fakat HTML5'in duyurulması (kullanılması) ile beraber etkinliği (önemi) artan (artacak olan) diğer teknolojilerin de modern web tarayıcıları tarafından desteklenmeye başlanmasıdır. (Örneğin; *WebGL*, *FileReader*, *Faster JavaScript*, *Geolocation API* gibi.)

Kısaca HTML5 dilini şöyle özetleyebiliriz:

- Sadeleştirilmiş, düzeltilmiş HTML4 ve XHTML dillerinin son sürümüdür.
- Daha fazla yapısal elemana, form elemanlarına ve yeni özelliklere sahiptir.
- Gelişmiş multimedia desteği sunmaktadır.
- İçerisinde yeni teknolojiler barındırır ya da önceden var olan teknolojilerin kullanımını sağlar.
- JavaScript ve CSS3 teknolojilerini daha etkili bir şekilde kullanmamızı sağlar.

HTML5'in nasıl ortaya çıktığı ile ilgili kısa bir bilgi verelim...

1999 yılında **W3C** (**W***ord* **W***ide* **W***eb* **C***onsortium*) tarafından **HTML 4.01** duyurulmuştur. HTML'in bu sürümünden kısa bir süre sonra XML 1.0 yapısı yayınlandı. W3C HTML dilini XML tabanlı yapmak için bu iki yapıyı birleştirdi ve 2001 yılında **XHTML 1.0** olarak duyurdu. 2003 yılında W3C tarafından **XForms 1.0** (*XHTML Extended Forms*) yayınlanarak var olan XHTML yapısı güçlendirilmek istendi. W3C yazarlarının düşüncesi XHTML yapısının geliştirilmesi ve ek teknolojilerle desteklenmesi şeklindeydi (*XHTML2+XForms+SVG+MathMl+RDFa*). Bu yapı tarayıcı üreticileri tarafından kabul görmediği için 2004 yılında **Apple**, **Mozilla Foundation** (*Firefox*) ve **Opera** tarayıcı üreticileri bir araya gelerek **WHATWG** (**W***eb* **H***ypertext* **A***pplication* **T***echnology* **W***orking* **G***roup*) isimli bir çalışma gurubu oluşturdular. Aslında HTML5'in doğuş hikayesi burada başlamıştır.

2006 yılında W3C konsorsiyumu XHTML dilini geliştirmekten vazgeçip HTML5'in gelişimine katılacağını duyurdu. 2008 yılında W3C tarafından **HTML5**'in (*First Public Working Draft*) ilk çalışma taslağı duyuruldu. W3C konsorsiyumu 2009 yılında **XHTML 2.0** çalışmalarını durduracağını açıklamıştır.

HTML5'İN YAPISI

HTML5'in temel elemanlar için getirdiği yeniliklere bakalım;

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>HTML5</title>
    </head>
    <body>
```

```
    </body>
</html>
```

Yeni DOCTYPE Bildirimi

```
<!DOCTYPE html>
```

Belge türünü ayarlamak için kullandığımız bu etiket, HTML5 ile daha kısa ve kullanışlı bir hale getirilmiştir. Örneğin; XHTML 1.0'da belge türü üç şekilde ayarlanabiliyordu ve tanımlama oldukça uzundu. HTML5, bu karmaşayı ortadan kaldırmış ve geriye doğru uyumluluğu (eski HTML sürümleri ile) korumuştur. Bu etiketin HTML5 ağaç yapısı içinde belgenin en başında tanımlanması zorunludur.

Şimdi önceki HTML sürümlerinde yapılan DOCTYPE tanımlamalarına bakalım.

XHTML 1.0 DOCTYPE (Transitional) tanımlaması:

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
```

XHTML 1.0 DOCTYPE (Strict) tanımlaması:

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
```

HTML 4.01 DOCTYPE (Strict) tanımlaması:

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01//EN”
“http://www.w3.org/TR/html4/strict.dtd”>
```

Dikkat ederseniz tanımlamalar oldukça uzun ve aslında Strict olan yapılar geriye doğru uyumluluğu desteklemiyor. Örneğin; sayfanızda XHTML 1.0 DOCTYPE (Strict) tanımlamasını kullanırsanız. Belge içerisinde sadece XHTML 1.0 sürümünde var olan (desteklenen) etiketleri ya da özellikleri kullanmanıza izin verilir. Strict tanımlaması yaparsanız önceki HTML sürümlerinde var olan fakat XHTML 1.0 tarafından desteklenmeyen etiket ya da özellikleri kullanmanıza izin verilmez. İşte bu durum, Strcit yapısının geriye doğru uyumlu olmadığı anlamına gelir.

Belge içerisinde `DOCTYPE` tanımlamasını yapmazsak, tarayıcı sayfayı **Quirks Mod**'da çalıştıracaktır. Tam da burada tarayıcı modları ile ilgili bilgi vermek istiyorum. Tarayıcı modları tarayıcının web sayfasını nasıl yorumladığını/yorumlayacağını gösterir. IE5/MAC sürümü ile beraber tarayıcılarda gösterilecek web sayfalarının

standartlara uyması beklenmiştir. Standartlara bağlı kalınmadan yazılan eski web sayfalarının tarayıcılar tarafından gösterilebilmesi için IE5 ile beraber Microsoft DOCTYPE kavramını ortaya atmıştır. Buna göre web belgesinin en başında DOCTYPE tanımlaması olan sayfalar standart modda, DOCTYPE tanımlaması olmayan (eski sayfalar) tuhaf modda çalışır. Diğer tarayıcı firmalarının da benimsemesi ile **Tarayıcı Modları** kavramı ortaya çıkmıştır.

Standard olarak üç tane tarayıcı modu bulunmaktadır. Şimdi bunları inceleyelim.

- **Quirks Mode:** DOCTYPE tanımlamasını yapmazsak tarayıcının sayfayı Quirks Mod'da çalıştıracağını söylemiştik. Bu durumda sayfanız oluşturulurken tarayıcı W3C standartlarına bağlı kalmaz. Sayfa görünümü ya da elemanların yerleşimi kullanılan tarayıcıya bağlı olarak değişiklik gösterebilir. DOCTYPE tanımlaması yapmamış iseniz; web sayfanız tarayıcının var olan eski kurallarına göre değerlendirilecek ve tarayıcıda oluşturulacaktır. Tarayıcı, Quirks modda çalıştığında farklı tarayıcılarda CSS özelliklerinin doğru olarak sayfa elemanlarına uygulanmasında farklı problemler çıkabilmektedir. Ayrıca tarayıcı Quirks Modda kurallara bağımlılık açısından daha esnek davranacaktır. Örneğin; DOCTYPE tanımlaması yapmazsanız IE6 ve alt sürümleri W3C kutu modeli yerine kendi kutu modelini kullanır.
- **Standart Mode:** DOCTYPE tanımlaması yaparak sayfamızın tarayıcı tarafından standart modda gösterileceğini belirtmiş oluruz. Standart modda tarayıcılar standartlara dayalı sayfamızı yorumlayacaktır. Standart modda tarayıcı sıkı bir denetim yapar ve sayfa görüntüsü oluşturur. HTML5 bildiriminde **no quirks mode** olarak adlandırılmıştır.
- **Almost Standart Mode:** Standart mod ile hemen hemen aynı olan bu mod; Opera, Safari, Chrome, Firefox ve IE8 tarafından desteklenmektedir. HTML5 bildiriminde **limited quirks mode** olarak adlandırılmıştır.

Yeni Character Set Bildirimi

```
<meta charset="utf-8">
```

Meta etiketi <head> etiketleri arasında bildirilir ve sayfa hakkında tarayıcılara ya da arama motorlarına bir takım bilgiler sunar. charset özelliği belge içerisinde kullanılacak karakter seti (kümesi) tanımlaması için kullanılır.

Önceki HTML sürümlerinde charset tanımlamasının nasıl yapıldığına bakalım.

```
<meta http-equiv="content-type" content="text/html;charset=utf-8">
```

`meta` etiketinin özelliklerine kısaca bakalım. (Standart Özellikler 2. Bölüm'de anlatılacaktır.)

- **`name:`** Meta etiketi içerisinde değer atanacak önceden tanımlı alt özellik isimlerini tanımlamak için kullanılır. Bu özellikle tanımlanan alt özelliğe değer atamak için `content` özelliği kullanılır.

  `name` özelliğine atanabilecek alt özellikler aşağıda listelenmiştir.

  `application-name:` Uygulama adını tanımlamak için kullanılır.

  `author:` Web sayfasını oluşturan kişi hakkında bilgi tanımlamak için kullanılır.

  `description:` Sayfa içeriği ile ilgili tanımlayıcı bir bilgi (açıklama) tanımlamak için kullanılır.

  `generator:` Sayfanın oluşturulduğu program (yazılım) ile ilgili bilgi tanımlamak için kullanılır.

  `keywords:` Arama motorları için virgülle ayrılmış anahtar kelimeler tanımlamak için kullanılır.

  Örnek:

  ```
  <meta name="keywords" content="HTML5, HTML5 Programlama, HTML5 JavaScript">
  ```

  Yukarıdaki bildirim web sitesi ile ilgili arama motorları için anahtar kelimeler tanımlar. Ayrıca **WHATWG MetaExtensions** ile tanımlanan aşağıdaki alt özellikleri de kullanabilirsiniz.

  `creator`, `googlebot`, `publisher`, `robots`, `slurp`, `viewport`

- **`http-equiv:`** `default-style`, `refresh` alt özelliklerini değer atamak için tanımlar. Belirtilen alt özelliklere `content` özelliği ile değerler atanır.

  `default-style:` Alternatif stil tanımlaması yapmak için kullanılır.

  `refresh:` Belirtilen süre sonunda sayfayı tekrar yüklemek ya da kullanıcıyı başka bir URL adresine yönlendirmek için bu alt özelliği kullanabilirsiniz.

  Örnek:

  ```
  <meta http-equiv="refresh" content="5; url=http://www.ibrahimcelikbilek.com" >
  ```

  Mevcut sayfa yüklendikten 5 sn sonra kullanıcı url ile belirtilen adrese yönlendirilecektir.

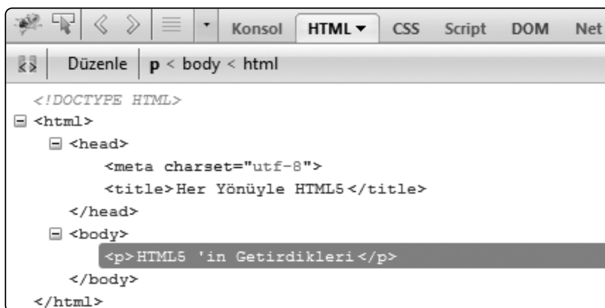
- **content:** `http-equiv` ya da `name` özelliklerinde tanımlanan alt özelliklere değer atamak için kullanılır.
- **charset (HTML5):** Sayfada kullanılan karakter seti (kümesi)'ni tanımlamak için kullanılır. HTML5 ile yeni tanımlanan bir özelliktir. Aslında UTF–8 kullanmak çoğu zaman yeterli olacaktır. Sayfanız için bir `charset` tanımlaması yapmazsanız sayfanızdaki karakter ya da sembollerin gösteriminde problem yaşayabilirsiniz. `charset` ile karakter setini tanımlamak aslında tarayıcıların karakter ya da sembolleri düzgün bir şekilde göstermeleri için kullanılan algoritmanın bir parçasıdır.

  HTML5 bildiriminde; meta etiketinin `scheme` özelliği ve `http-equiv` özelliğinin `content-type`, `content-language`, `set-cookie` alt özelliklerinin kullanılması önerilmemektedir.

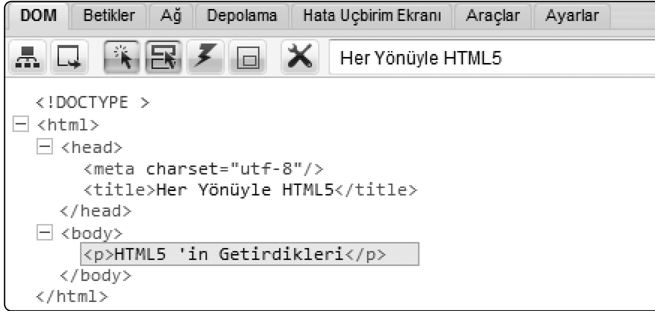
HTML5 Söz Dizimi Kuralları

HTML5, XHTML gibi XML temelli olmadığı için programcılara çok gevşek bir söz dizimi yapısı sunar. Örneğin; bir HTML5 sayfası oluştururken `html`, `head`, `body` etiketlerini belirli şartlar altında kullanmayabilirsiniz. Bu durumda sizin yazmadığınız `html`, `head`, `body` etiketleri tarayıcı tarafından sayfaya dahil edilecektir. Ancak bu durum kullanışlı değildir. Çünkü bu durumda IE'de problemler yaşabiliriz. Aşağıdaki uygulamamızı HTML5'in ne kadar esnek bir söz dizimi olduğunu göstermek için yapıyorum. Biz uygulamalarımızda `html`, `head`, `body` elemanlarını HTML5 temel ağaç yapısı içerisinde her zaman kullanacağız. Aşağıdaki örneği farklı tarayıcılarda çalıştırıp sonuca bakalım.

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>Her Yönüyle HTML5</title>
<p>HTML5'in Getirdikleri</p>
```



Firefox (Firebug)



Opera (Dragonfly)

Dikkat ederseniz Firefox ve Opera sayfayı gösterirken; `html`, `head`, `body` elemanlarını sayfaya dahil etti. Bunun yanında IE8 yazılmayan `html`, `head`, `body` etiketlerinin sadece açılış taglarını oluşturulur.

HTML5'in söz dizimi kurallarına bakalım...

1. Açılan tüm etiketler kapatılmalıdır. Bazı etiketler bir içerik barındırmaz, sakladıkları içerikler etiketin bir özelliği ile tanımlanır. Bu durumda etiket açılış tagında **/** karakteri ile kapatılır. Burada şunu belirtmek isterim. Eğer elemanımız **void elements** (açılış kapanış tagları arasında bir text (*metin*) ya da başka bir eleman bulunmayan, yani sadece açılış tagı (*etiketi*) bulunan elemanlar; örneğin; `area`, `base`, `br`, `col`, `command`, `embed`, `hr`, `img`, `input`, `keygen`, `link`, `meta`, `param`, `source`) ise bu durumda / karakterinin, bu elemanlar üzerinde bir etkisi olmayacaktır. Yani bu karakteri kullanmayabilirsiniz. Fakat elemanımız **foreign elements** (namespace-isim alanı içeren elemanlar) ise bu durumda açılış tagı **/** karakteri ile kapatılmalıdır.

   Örnek kullanımlar:

   `<a>Link` (Yanlış, kapanış etiketi yok.)

   `<p>Açıklama<p>` (Yanlış, kapanış etiketi yok.)

   `<div>Div elemanı içeriği</div>` (Doğru.)

   `<span></span>` (Doğru.)

   `<meta charset="utf-8">` ya da `<meta charset="utf-8"/>` (İkisi de doğru.)



Tavsiye: Sadece açılış etiketi bulunan tüm elemanları kapatmadan önce / karakterini kullanınız. Örneğin; `<img src="ana.jpg" alt="ana_resim" />`

2. İç içe girmiş, bir biri içerisine yuvalanmış elemanlarda elemanların kapanış sırasına dikkat edilmelidir. İlk önce açılan etiket, en sonda kapatılmalıdır.

```
<div>Div <em>elemanı <span>içeriği</span></em></div>
```
(Doğru)

```
<div>Div <em>elemanı <span>içeriği</em></span></div>
```
(Yanlış)

3. Etiket içerisinde kullanılan özellik isimleri büyük ya da küçük harfle yazılabilir. Özelliğin değeri kendisi ile aynı ismi taşıyorsa sadece özellik adı yazılabilir. Özellik değerleri eğer bir boşluk içermiyorsa tırnak içerisine alınmayabilir. Ya da isterseniz özellik isimlerini tek tırnak ya da çift tırnak içerisine alabilirsiniz.

Örnek kullanımlar:

```
<input type="checkbox" checked />
```

`checked` özelliğinin değeri `"checked"` olduğundan yazılmadı. Sadece özellik adı yazıldı.

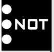
```
<img src="1.jpg" alt='1resim' />
```

`alt` özelliğine atanan metin tek tırnak içerisine alınarak yazıldı.

```
<div class=header></div>
```

`class` özelliğine atanan değer tırnak içerisine alınmadan yazıldı.

Yukarıdaki kullanımlar HTML5 için doğru olarak kabul edilir.



Tavsiye: Etiket ismi, etiket içinde kullanılan özellik ismi ya da özelliğe atanan değerleri küçük harfle yazınız. Değerleri çift tırnak içerisine alarak özelliklere atayınız.

Temiz (okunabilir), tutarlı (az hatalı) kodlar oluşturmak için tavsiyelere uymanızı öneririm.

HTML5 İçin Tarayıcı Desteği

Peki, HTML5'i tarayıcılar, ne kadar destekliyor?

Aşağıda ayrıntılı bir şekilde anlatacağım ama şunu bilmenizi isterim. Yeni nesil tarayıcılar (Modern web tarayıcılarının son sürümleri), HTML5'in yeni getirdiği teknolojileri, elemanları ve özellikleri destekleme konusunda hızlı adımlar atmaktadırlar. Bu adımlar sonucunda tarayıcıların son sürümleri (*IE9*, *Firefox 4*, *Safari 5*, *Opera 11.11*) HTML5'i büyük ölçüde destekler duruma gelmişlerdir. HTML5'e tarayıcılar tarafından verilen desteği sadece yeni elemanlar ya da özelliklere verilen destek olarak düşünmemek gerekir. Bu destek; CSS3, JavaScript eklentileri, DOM

(*Document Object Model*) ve HTML5'in duyurulması (kullanılması) ile beraber etkinliği (önemi) artan (artacak olan) yeni ya da var olan teknolojileri de bir paket olarak içermektedir. Bu nedenle aşağıda HTML5 teknolojisinin getirdiği ya da HTML5 ile beraber kullanımı artan/kullanımı giren tüm teknolojilerin tarayıcılar tarafından ne ölçüde desteklendiğine bakmış olacağız.

Tarayıcı Görüntüleme Motoru (Layout Engine) ve JavaScript Motoru (JavaScript Engine)

Tarayıcı görüntüleme motoru, web sayfasının içeriğini tarayıcı ekranında oluşturmak/göstermekle sorumludur. Tarayıcılar modüler sistemi benimsemişlerdir. Bu şu anlama gelir tarayıcı yapısı genel olarak kullanıcı arayüzü ve iş yapan arabirimler olmak üzere ikiye ayrılabilir. Web tarayıcısı geliştiricileri aslında görüntüleme motoru temelli tarayıcı oluştururlar. Tarayıcılar kendi oluşturdukları görüntüleme motorlarını kullanırlar ve bunları güncellerler (Aslında görüntüleme motorlarını güncellediklerinde belirtilen tarayıcı için yeni bir sürüm ortaya çıkmış olur). Bu görüntüleme motorlarına dayalı uygulamalar geliştirirler. Bir tarayıcının yeni teknolojileri desteklemesi; görüntüleme motorunun belirtilen teknolojileri desteklemesi anlamına gelir. Tarayıcılarda kullanıcı arayüzleri sadece görüntüleme motorunun işlediği verileri kullanıcıya gösteren ve kullanıcı ile iletişimde bulunan bir yapı olarak tanımlanabilir.

Tarayıcılar, web belgesi içerisindeki JavaScript kodlarını yorumlamak ve işlevlerini yerine getirmek (derlemek) için kendi oluşturdukları JavaScript motorlarını (*JavaScript Engine*) kullanılırlar (Bu JavaScript motorlarının bir kısmı açık kaynak kodludur). JavaScript yorumlayıcıları kullandıkları bellek miktarı ya da diğer özellikleri ile tarayıcı hızını etkileyen en önemli faktörlerin başında gelir.

Tarayıcıların kullandıkları görüntüleme motorları, JavaScript yorumlayıcıları aşağıda listelenmiştir.

Firefox

Layout Engine Adı: Gecko

Gecko Versiyonu	Tarayıcı Sürümü
Gecko 1.7	Firefox 1.0
Gecko 1.8	Firefox 1.5
Gecko 1.8.1	Firefox 2
Gecko 1.9	Firefox 3
Gecko 1.9.2	Firefox 3.6
Gecko 2	Firefox 4 (JS Yorumlayıcısı JägerMonkey)

Opera

Layout Engine Adı: Presto

Presto Versiyonu	Tarayıcı Sürümü
1.0	Opera 7.0
2.0	Opera 9.0
2.1	Opera 9.5
2.1.1	Opera 9.6
2.2.15	Opera 10.0, Opera 10.1
2.5.24	Opera 10.5
2.6.30	Opera 10.6
2.7.62	Opera 11 (JS Yorumlayıcısı Carakan)

Internet Explorer

Layout Engine Adı: Trident

Trident Versiyonu	Tarayıcı Sürümü
4.0	IE8
5.0	IE9 (JS Yorumlayıcısı Chakra) Diğer alt sürümleri için modüler bir isim belirtilmemiştir.

Safari

Layout Engine Adı: WebKit

WebKit Versiyonu	Tarayıcı Sürümü
526.12.2 , 528.1.1 , 528.16 , 528.17	Safari 4.0
530.17	Safari 4.1
530.19.1	Safari 4.0.2
531.9.1	Safari 4.0.3
531.21.10	Safari 4.0.4
531.22.7	Safari 4.0.5
533.16	Safari 5.0
533.18.5	Safari 5.0.2
533.19.4	Safari 5.0.3 (JS Yorumlayıcısı Nitro)

Yukarıdaki liste de Windows işletim sistemi için üretilen Safari sürümleri ve WebKit versiyonları yazılmıştır.

Aşağıdaki tablolarda tarayıcıların tüm versiyonlarının; HTML5 etiket, özellik ve teknolojilerini destek durumlarını daha iyi göstermek için tarayıcıların Layout Engine versiyon numaraları kullanılmıştır.

Tablolardaki yeni eleman, özellik ya da teknolojinin karşısında bulunan **Layout Engine** sürümü ve daha sonra çıkan sürümleri ilgili eleman, özellik ya da teknolojiyi destekler. Ayrıca aşağıdaki tarayıcı destek durumları bu kitap yazıldığı tarih itibariyle var olan durumdur.

HTML5 İLE YENİ TANIMLANAN ELEMANLARA VE ÖZELLİKLERE TARAYICILARIN VERDİĞİ DESTEK

Yeni Yapısal Elemanlar

<table>
<tr><th>Eleman Adı</th><th>Internet Explorer</th><th>Firefox</th><th>Opera</th><th>Safari</th></tr>
<tr><td><section></td><td rowspan="9">Trident 5.0</td><td rowspan="9">Gecko2</td><td rowspan="10">Presto 2.7</td><td rowspan="7">WebKit 533</td></tr>
<tr><td><nav></td></tr>
<tr><td><article></td></tr>
<tr><td><aside></td></tr>
<tr><td><hgroup></td></tr>
<tr><td><header></td></tr>
<tr><td><footer></td></tr>
<tr><td><figure>
<figcaption></td><td>Nightly Build*
eklentisi İle</td></tr>
<tr><td><mark></td><td rowspan="2">Win7 için yok</td></tr>
<tr><td><progress></td><td>Yok</td><td>Yok</td></tr>
<tr><td><time></td><td>Yok</td><td>Yok</td><td>Yok</td><td>Yok</td></tr>
<tr><td><meter></td><td>Yok</td><td>Yok</td><td>Presto 2.7</td><td>Desteği var</td></tr>
<tr><td><command></td><td>Yok</td><td>Yok</td><td>Yok</td><td>Yok</td></tr>
<tr><td><details></td><td>Yok</td><td>Yok</td><td>Yok</td><td>Kısmen</td></tr>
<tr><td><summary></td><td>Yok</td><td>Yok</td><td>Yok</td><td>Kısmen</td></tr>
<tr><td><ruby>, <rt>,
<rb></td><td>Desteği Var</td><td>Yok</td><td>Yok</td><td>WebKit 533</td></tr>
</table>

Yeni Media Elemanları

<table>
<tr><th>Eleman Adı</th><th>Internet Explorer</th><th>Firefox</th><th>Opera</th><th>Safari</th></tr>
<tr><td><audio></td><td rowspan="2">Trident 5.0</td><td rowspan="2">Gecko1.9.1
(Firefox 3.5)</td><td rowspan="2">Presto 2.5</td><td rowspan="2">WebKit 525
(Safari 3.0)</td></tr>
<tr><td><video></td></tr>
<tr><td><source></td><td></td><td></td><td></td><td></td></tr>
<tr><td><embed></td><td>IE 3.0</td><td>Gecko 1.7</td><td>Presto 1.0</td><td>Desteği var</td></tr>
</table>

Canvas Elemanı

Eleman Adı	Internet Explorer	Firefox	Opera	Safari
<canvas>	Trident 5.0	Gecko1.9.2	Presto 2.0	Desteği var

Belirtilen tarayıcıların canvas elemanın hangi özellik ve yöntemlerini desteklediği Bölüm 6'da ayrıntılı bir şekilde anlatılacaktır.

Yeni Form Elemanları

<table>
<tr><th>Eleman Adı</th><th>Internet Explorer</th><th>Firefox</th><th>Opera</th><th>Safari</th></tr>
<tr><td><datalist></td><td rowspan="3">Yok</td><td rowspan="2">Gecko 2.0</td><td rowspan="2">Presto 2.0</td><td rowspan="2">Yok
Nightly Build*
eklentisi ile</td></tr>
<tr><td><output></td></tr>
<tr><td><keygen></td><td>Desteği var</td><td>Presto 1.0</td><td>Desteği var</td></tr>
</table>

Input Etiketinin Type Özelliğine Atanabilecek Yeni Değerler

<table>
<tr><th>Değer Adı</th><th>Internet Explorer</th><th>Firefox</th><th>Opera</th><th>Safari</th></tr>
<tr><td>search</td><td rowspan="13">Yok</td><td rowspan="4">Gecko 2.0</td><td rowspan="2">Presto 2.7</td><td>Desteği var</td></tr>
<tr><td>tel</td><td rowspan="3">WebKit 528</td></tr>
<tr><td>url</td><td rowspan="10">Presto 2.0</td></tr>
<tr><td>email</td></tr>
<tr><td>datetime</td><td rowspan="9">Yok</td><td rowspan="7">Win7 için yok</td></tr>
<tr><td>date</td></tr>
<tr><td>month</td></tr>
<tr><td>week</td></tr>
<tr><td>Time</td></tr>
<tr><td>datetime-local</td></tr>
<tr><td>Number</td></tr>
<tr><td>range</td><td>Desteği var</td></tr>
<tr><td>color</td><td>Presto 2.7</td><td>Win7 için yok</td></tr>
</table>

Input Etiketi için Yeni Özellikler

<table>
<tr><th>Özellik Adı</th><th>Internet Explorer</th><th>Firefox</th><th>Opera</th><th>Safari</th></tr>
<tr><td>autocomplete</td><td>Desteği var (IE 8.0)</td><td>Gecko1.9.1 (Firefox 3.5)</td><td rowspan="3">Presto 2.0</td><td rowspan="2">Desteği var (Safari 4.0)</td></tr>
<tr><td>autofocus</td><td rowspan="2">Yok</td><td rowspan="2">Gecko 2.0</td></tr>
<tr><td>form</td><td>Nightly Build* eklentisi ile</td></tr>
<tr><td>height and width</td><td>Desteği var (IE 8.0)</td><td>Yok</td><td>Presto 2.5.24</td><td>Desteği var (Safari 4.0)</td></tr>
<tr><td>list</td><td rowspan="7">Yok</td><td>Gecko 2.0</td><td>Presto 2.0</td><td>Yok</td></tr>
<tr><td>min, max, step</td><td>Yok</td><td>Presto 2.0</td><td>min, max Nightly Build* eklentisi İle
step WebKit 528</td></tr>
<tr><td>multiple</td><td>Gecko1.9.1 (Firefox 3.5) Sadece type="file" İçin</td><td>Presto 2.7</td><td rowspan="3">WebKit 528</td></tr>
<tr><td>novalidate</td><td rowspan="3">Gecko 2.0</td><td>Presto 2.7</td></tr>
<tr><td>pattern</td><td>Presto 2.0</td></tr>
<tr><td>placeholder</td><td>Presto 2.7</td><td>Desteği var (Safari 4.0)</td></tr>
<tr><td>required</td><td rowspan="6"></td><td rowspan="6"></td><td>Presto 2.0</td><td>WebKit 528</td></tr>
<tr><td>formtarget</td><td rowspan="5">Presto 2.7</td><td rowspan="4">Nightly Build* eklentisi ile</td></tr>
<tr><td>formaction</td></tr>
<tr><td>formmethod</td></tr>
<tr><td>formenctype</td></tr>
<tr><td>formnovalidate</td><td>WebKit 528</td></tr>
</table>

Diğer Specifications (W3C Bildirimleri) ve Teknolojilere Tarayıcıların Verdiği Destek

Özellik Adı	Internet Explorer	Firefox	Opera	Safari
Selectors API Level1	IE8	Firefox 3.5	Opera 10	Safari 3
Web Storage			Opera 10.5	
Web SQL Database	Yok	Yok	Opera 10.5	Safari 3.2
WebSockets	Yok	Firefox 4	Opera 11	Safari 5
Web Workers	Yok	Firefox 3.5	Opera 10.6	Safari 5
Geolocation API	Yok			
Offline Web Applications	Yok			Safari 4
WebGl	Yok	Firefox 4	Yok	Nightly Build* eklentisi ile
Drag-And-Drop	Kısmen	Firefox 3.5	Yok	Desteği var

HTML5 ile beraber gelen Standart (Ortak) özellikler ve elemanlar için tanımlanan diğer yeni özellikler 2. Bölümde anlatılacaktır.

HTML5 Dili İçerisindeki Tüm Etiketlerin Listesi

Etiket	Durum
<a>	
<abbr>	
<acronym>	HTML5 tarafından desteklenmiyor.
<address>	
<applet>	HTML5 tarafından desteklenmiyor.
<area>	
<article>	Yeni (HTML5)
<aside>	Yeni (HTML5)
<audio>	Yeni (HTML5)

<b>	
<base>	
<basefont>	HTML5 tarafından desteklenmiyor.
<bdo>	
<big>	HTML5 tarafından desteklenmiyor.
<blockquote>	
<body>	
<button>	
<canvas>	Yeni (HTML5)
<caption>	
<center>	HTML5 tarafından desteklenmiyor.
<cite>	
<code>	
<col>	
<colgroup>	
<command>	Yeni (HTML5)
<datalist>	Yeni (HTML5)
<dd>	
<del>	
<details>	Yeni (HTML5)
<dfn>	
<dir>	HTML5 tarafından desteklenmiyor.
<div>	
<dl>	
<dt>	
<em>	
<embed>	Yeni (HTML5)

<fieldset>	
<figcaption>	Yeni (HTML5)
<figure>	Yeni (HTML5)
<font>	HTML5 tarafından desteklenmiyor.
<footer>	Yeni (HTML5)
<form>	
<frame>,<frameset>	HTML5 tarafından desteklenmiyor.
<h1>..<h6>	
<head>	
<header>	Yeni (HTML5)
<hgroup>	Yeni (HTML5)
<hr>	
<html>	
<i>	
<iframe>	
<img>	
<input>	
<ins>	
<keygen>	Yeni (HTML5)
<kbd>	
<label>	
<legend>	
<link>	
<map>	
<mark>	Yeni (HTML5)
<menu>	
<meta>	

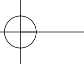
<meter>	Yeni (HTML5)
<nav>	
<noframes>	HTML5 tarafından desteklenmiyor.
<noscript>	
<object>	
<optgroup>	
<option>	
<output>	Yeni (HTML5)
<p>	
<param>	
<pre>	
<progress>	Yeni (HTML5)
<q>	
<rp>	Yeni (HTML5)
<rt>	Yeni (HTML5)
<ruby>	Yeni (HTML5)
<s>	
<samp>	
<script>	
<section>	Yeni (HTML5)
<select>	
<small>	
<source>	Yeni (HTML5)
<span>	
<strike>	HTML5 tarafından desteklenmiyor.
<strong>	
<style>	

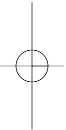
<sub>	
<summary>	Yeni (HTML5)
<sup>	
<table>	
<tbody>	
<td>	
<textarea>	
<tfoot>	
<th>	
<thead>	
<time>	Yeni (HTML5)
<title>	
<tr>	
<tt>	HTML5 tarafından desteklenmiyor.
<u>	HTML5 tarafından desteklenmiyor.
<var>	
<video>	Yeni (HTML5)
<wbr>	Yeni (HTML5)
<!-- HTML Comment -->	
<!DOCTYPE>	

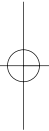
NOT

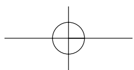
HTML5 dili içerisine eklenen yeni etiketler ilerleyen bölümlerde uygulamalı olarak anlatılacaktır. Fakat HTML diline yeni başlıyorsanız, HTML5 yapısında bulunan ve önceki HTML sürümlerinden gelen eski etiketlerle ilgili bilgi almak için; KODLAB'tan çıkan XHTML ve CSS isimli kitabımdan faydalanabilirsiniz.











YENİ ELEMANLAR VE ÖZELLİKLER

2

Bu bölümde HTML5 ile beraber yeni tanımlanan elemanlar ve özelliklere bakacağız. Fakat daha önce içerik modeli ve tüm elemanların sahip oldukları ortak özellikleri inceleyelim.

HTML5 dilinde elemanların saklayabilecekleri içerikler guruplandırılmış ve **İçerik Modeli** (*Content Models*) olarak adlandırılmıştır. İçerik modeli elemanların ne çeşit bir içeriğe sahip olabileceklerini tanımlamak için kullanılır. Bir eleman birden fazla içerik türünü destekleyebilir.

HTML5 tarafından tanımlanan içerik türleri şöyledir:

Content Type	Açıklama
Embedded	Dış kaynaklı ve ya başka bir etiketleme dili ile ya da programatik olarak (Örneğin; JavaScript ile Canvas elemanı kullanımı) tanımlanan içerikler. Bu içerik türünü kullanan elemanlara örnek; `<audio>`, `<canvas>`, `<embed>`, `<iframe>`, `<img>`, `<math>`, `<object>`, `<svg>`, `<video>`
Flow	Metin, başka bir eleman ya da gömülü olarak tanımlanan içerikler. Bu içerik türünü kullanan elemanlara örnek; `<a>`, `<abbr>`, `<address>`, `<article>`, `<aside>`, `<audio>`, `<b>`, `<bdo>`, `<blockquote>`, ` `, `<button>`, `<canvas>`, `<cite>`, `<i>`, `<iframe>`, `<img>`, `<input>`, `<script>`, `<section>`, `<select>`, `<small>`
Heading	Bir bölümün başlığını tanımlayan içerikler. Bu içerik türünü kullanan elemanlara örnek; `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<hgroup>`

Sectioning	<article>, <aside>, <nav>, <section> elemanlarının içerik türüdür.
Metadata	<base>, <command>, <link>, <meta>, <noscript>, <script>, <style>, <title> elemanlarının içerik türüdür. Sayfaların görselliğini ve davranışlarını değiştirebilen içeriklerdir.
Phrasing	Paragrafları oluşturan metin ya da başka bir eleman olarak tanımlanan içerikler. Bu içerik türünü kullanan elemanlara örnek; <span>, <strong>, <sub>, <sup>, <svg>, <textarea>, <time>, <var>, <video>, <wbr>, <kbd>, <keygen>, <label>, <mark>, <math>, <meter>...
Interactive	Kullanıcı ile etkileşim için tanımlanmış interaktif içerikler. Bu içerik türünü kullanan elemanlara örnek; <a>, <button>, <details>, <embed>, <iframe>, <keygen>, <label>, <select>, <textarea>

Standart Özellikler

HTML5'de her eleman aşağıdaki ortak özelliklere sahiptir. Şimdi bu özelliklerin neler olduğuna bakalım.

- accesskey
- class
- contenteditable [HTML5]
- contextmenu [HTML5]
- dir
- draggable [HTML5]
- dropzone [HTML5]
- id
- lang
- spellcheck [HTML5]
- style
- tabindex
- title
- hidden [HTML5]

ACCESSKEY

Alabileceği Değer: {Karakter}

HTML elemanına klavye kısayolu tanımlamak için kullanılır.

CLASS

Aldığı Değerler: {Sınıf Seçici Adı [*Birden fazla sınıf seçici adı yazılabilir]}

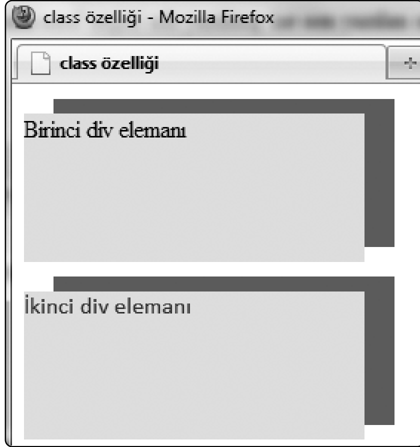
Oluşturulmuş bir sınıf seçiciyi HTML elemanına atamak (uygulanmasını sağlamak) için kullanılır.

Örnek:

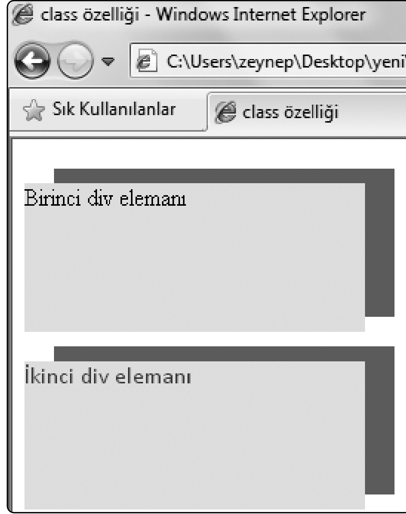
```
<!DOCTYPE HTML>
<html>
<head>
<meta  charset="utf-8" />
<title>class özelliği</title>
<style type="text/css">
/*Bu Kitap boyunca uygulamalarda CSS ve JavaScript dilleri ile ilgili kısa ve uygulamaya
yönelik bilgiler verilecektir.*/
/*box-shadow özelliği; html elemanı için tanımlı olan kutuya gölge efekti vermek için kullanılır
ve CSS3  ile tanımlanan bir özelliktir.
Söz Dizimi:
box-shadow:inset <offset-x><offset-y><blur-radius><spread-radius><color> ;
inset,blur-radius,spread-radius,color ; isteğe bağlı
<offset-x> <offset-y>:gerekli*/
              .shadow{
                      margin-top:20px;
                      width:230px;
                      height:100px;
                      background-color:khaki;
                      /*Firefox  3.5 ve 3.6 için*/
                      -moz-box-shadow:20px -10px crimson;
                      /*IE9 ,  Opera 10.5 , Firefox 4.0  için  CSS3 Özelliği*/
                      box-shadow:20px -10px  crimson;
                      /*Safari için*/
                      -webkit-box-shadow:20px -10px  crimson;
                      /*IE9 alt sürümleri için*/
filter:progid:DXImageTransform.Microsoft.DropShadow(OffX=20,
OffY=-10,     Color='crimson', Positive='true');
                      }
              .renk{
                      font-family:calibri;
                      color:blue;
              }
</style>
</head>
<body>
<p class="shadow">Birinci div elemanı</p>
<div class="renk shadow">İkinci div elemanı</div>
</body>
</html>
```

Eğer `class` özelliğine birden fazla sınıf seçici adı yazılmış ise, son yazılan sınıf seçici en öncelikli olur.

Sayfamızın görüntüsü:



Firefox 3.6 ekran görüntüsü



IE8 ekran görüntüsü

CONTENTEDITABLE [HTML5]

Alabileceği Değerler: `{true, false}`

Eleman içeriğinin düzenlenebilir olup olmadığını ayarlamak için kullanılır.

- **`true`** değeri atanmışsa eleman içeriği kullanıcı tarafından düzenlenebilir/değiştirilebilir.
- **`false`** değeri atanmışsa eleman içeriği kullanıcı tarafından düzenlenemez.

Eğer bu özellik bir elemana atanmışsa kalıtsal olarak elemanın içinde bulunan alt elemanlara geçer. Bu özelliği değeri ile beraber kullanmak şarttır.

Örneğin;

`<div contenteditable></div>` şeklindeki bir kullanım yanlış olacaktır.

Doğrusu;

`<div contenteditable="true" ></div>` şeklinde olmalıdır.

Örnek:

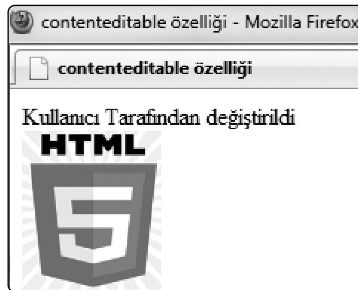
```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset=”utf-8” />
    <title> contenteditable özelliği</title>
    <style type=”text/css”>

    </style>
</head>
<body>
    <div contenteditable=”true”>
        Düzenlenebilir İçerik</div>
    <div contenteditable=”true”>
        <img alt=”düzenle” src=”html5.png” />
    </div>
</body>
</html>
```

Dikkat ederseniz sayfada bulunan div elemanları için `contenteditable` özelliği `true` değeri ile tanımlanmıştır. Bu durumda iki div elemanının da içerikleri kullanıcı tarafından değiştirilebilir. Burada şuna dikkatinizi çekmek istiyorum. İkinci div elemanının içerisinde bulunan `img` elemanı, bu özelliği kalıtsal olarak almaktadır. Tüm tarayıcılar `img` elemanının kullanıcı tarafından silinmesine izin verir, fakat sadece IE ve Firefox `img` elemanının sayfa üzerindeki boyutlarını değiştirmemize izin verecektir.



Sayfanın normal (varsayılan) ekran görüntüsü



Firefox 3.6 ekran görüntüsü

Tarayıcı Desteği: Internet Explorer 5.5 +, Firefox 3+, Opera 9.0+, Safari (Desteği var.)



+ karakteri belirtilen sürüm ve daha sonra çıkan sürümler anlamına gelmektedir.

CONTEXTMENU [HTML5]

Alabileceği Değer: `{menu elemanı id değeri}`

Bir elemanı menu elemanı ile ilişkilendirmek için kullanılır.

Tarayıcı Desteği: Internet Explorer (Yok), Firefox (Yok), Opera (Yok), Safari (Yok)

DIR

Alabileceği Değerler: `{ltr, rtl}`

HTML elemanının içerdiği metnin yazılış yönünü ayarlamak için kullanılır. `ltr` ile `rtl` değerlerinden birini alır.

DRAGGABLE [HTML5]

Alabileceği Değerler: `{true, false, auto}`

Bir HTML elemanının sürüklenebilir olup olmayacağını ayarlamak için kullanılır.

- `true` değeri atanmışsa kullanıcı mouse ile elemanının konumunu değiştirebilir. (Eleman sürüklenebilir)
- `false` değeri atanmışsa eleman sürüklenemez.
- `auto` değeri atanmışsa ve yukarıdaki değerler kullanılmamış ise tarayıcı kendi varsayılan değerini kullanılır.

Tarayıcı Desteği: Internet Explorer (Yok), Firefox 3.5+, Opera (Yok), Safari 5.0+

DROPZONE

Alabileceği Değerler: `{copy, move, link }`



Bu özellik, **Drag and Drop** isimli bölümde ayrıntılı bir şekilde anlatılacaktır.

Sürükle-bırak işlemlerinde bırakma anındaki tarayıcı davranışını tanımlamak için kullanılır.

- `copy` değeri atanırsa sürükleme işleminin bittiği yerde sürüklenen datanın kopyası oluşturulur.
- `move` değeri atanırsa sürüklenen datayı sürükleme işleminin bittiği yere taşır.
- `link` değeri atanırsa sürükleme işleminin bittiği yerde sürüklenen dataya bir link oluşturur.

Tarayıcı Desteği: Yok.

ID

Alabileceği Değer: {`Bir HTML elemanın id özelliğine atanan değer`}

HTML elemanlarına benzersiz bir isim vermek için kullanılır. Aslında `id` özelliği ile HTML elemanlarına bir kimlik numarası verilir. JavaScript ile `id` özelliğine değer atanmış yani bir kimliği olan elemanlara ulaşabilir ve çalışma anında bu elemanın stil ve yapısal özelliklerini değiştirebiliriz.

Örnek:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8" />
    <title>id özelliği</title>
    <script type="text/javascript">
        var goster = function (){
            var yeni_text = document.createTextNode("id özelliği");
            /* Yeni bir text (metin) düğümü oluşturmak için createTextNode() metodunu
              kullandık.
              *Kullanımı:
                *createTextNode(String):String (Bu gösterim Metodun hangi tip değer
                aldığını ve metodun geri döndürdüğü değer türünü tanımlar.)
                *Dom Level 1,2
            */
            var div = document.getElementById("icerik");
            /* id özelliğine "icerik" değerini almış div elemanının referansını
              aldık. getElementById() metodu belirtilen id değerini almış elemanı
              obje(Nesne) olarak döndürür.
              *Kullanımı:
                *getElementById(String):HTMLElement
                *Dom Level 1,2
              */
```

```
                div.appendChild(yeni_text);
               /* yeni_text isimli metin düğümünü appendChild() metodu ile div elemanı
                  içerisine ekledik.
                  *Kullanımı:
                     *appendChild(Node):Node
                     *Dom Level 1,2
                  */
          }
     </script>
     <style type="text/css">
          #icerik
          {
               height: 50px;
               width: 200px;
               background-color: lightblue;
               overflow:hidden;
          }
     </style>
</head>
<body>
     <div id="icerik" onmouseover="goster();">
     </div>
</body>
</html>
```

`id` özelliğine `"icerik"` değeri almış elemanın Mouse ile üzerine gelindiğinde yandaki ekran görüntüsü oluşur.



Firefox 3.6 ekran görüntüsü

LANG

Alabileceği Değer: `{Dil Kodu}`

HTML elemanlarının içerdikleri metinlerin ya da özelliklerine aldıkları değerlerin dilini ayarlamak için kullanılır. Örneğin; Türkçe: `tr`, Almanca: `de`, İngilizce: `en`

SPELLCHECK [HTML5]

Alabileceği Değerler: `{true, false}`

Eğer `true` değeri atanmışsa eleman içindeki metinde dilbilgisi ve yazım hataları kontrol edilir. `false` değeri atanmışsa kontrol edilmez. Bu özelliği düzenlenebilir

içeriğe sahip elemanlarda kullanabilirsiniz. Örneğin; `textarea`, `input` ya da `contenteditable="true"` değerini almış diğer elemanlar.

Tarayıcı Desteği: Internet Explorer (Yok), Firefox 2+, Opera (Yok), Safari (Yok)

STYLE

Alabileceği Değerler: `{CSS tanımlamaları(Özellik:Değer)}`

Bu özelliği kullanarak HTML elemanlarına satır içi CSS kodları yazabilirsiniz. Bu CSS kodları `style` özelliği ile yazıldıkları HTML elemanına uygulanır. `Style` özelliğine yazılan CSS kodları bir elemanı hedef alan CSS kodları içinde en öncelikli tanımlamalar olacaktır.

Örnek:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8" />
    <title>style özelliği</title>
    <style type="text/css">
        p
        {
            color: Maroon; /*color tanımlaması, p elemanına uygulanmaz.
            Çünkü öncelikli olan color tanımlaması style özelliğine yazılan tanımlamadır.*/
            width: 100px;
        }
    </style>
</head>
<body>
    <p style="background-color: khaki; color:hsl(360,100%,50%)">
        Her Yönüyle HTML5
    </p>
</body>
</html>
```

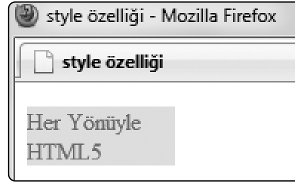
`p` elemanına atanan CSS özelliklerine görsel olarak inceleyelim.

```
element.style {
    background-color: #f0e68c;
    color: #ff0000;
}

p {
    color: #800000;
    width: 100px;
}
```

p elemanına atanan CSS özellikleri

Ekran görüntüsüne bakalım.



Firefox 3.6 ekran görüntüsü

TABINDEX

Alabileceği Değer: `{Sayı}`

HTML elemanı için sekme sırasını ayarlar.

TITLE

Alabileceği Değer: `{Metin}`

HTML etiketine bilgi ve açıklama ekleyebilirsiniz.

HIDDEN [HTML5]

Alabileceği Değer: `{hidden}`

HTML elemanını gizlemek için kullanılır.

Tarayıcı Desteği: Internet Explorer (Yok), Firefox 4.0, Opera (Presto/2.7.7), Safari (Nightly Build* ile)

HTML5'İN GETİRDİĞİ YENİ ELEMANLAR

HTML5 yapısında bulunan yeni elemanlara bakalım.

YAPISAL ELEMANLAR

Bir web sayfası tasarlarken muhtemelen sayfanız kabaca bir üst başlık alanı (sitenin tanıtımı için), bir içerik alanı ve yine içerik alanı içerisinde değişik alt başlıklar (bölümler), bir alt başlık ve bir ya da daha fazla navigasyon (yönlendirme) alanlarından oluşacaktır (Değişik site tasarımları olabilir. Burada genel yaklaşımdan söz edilmektedir). Sitenizin yapısını tasarlarken HTML5 ile yeni gelen yapısal elemanları kullanabilirsiniz. Bu elemanların kullanılması; kod okunabilirliğini arttırır, anlamlı eleman guruplar oluşturulmasını sağlar ve CSS'i daha etkili bir şekilde kullanmanıza yardımcı olur.

<HEADER>

Başlık elemanı, bu elemanı kullanarak sayfa başlık alanı (sayfa hakkında bilgiler içeren bölüm), bölüm veya alt bölüm başlığı ya da yönlendirme alanları için başlık oluşturabilirsiniz. Genelde `h1`, `h2`, `h3` .. `h6` elemanlarını ya da `hgroup` elemanını içerir.

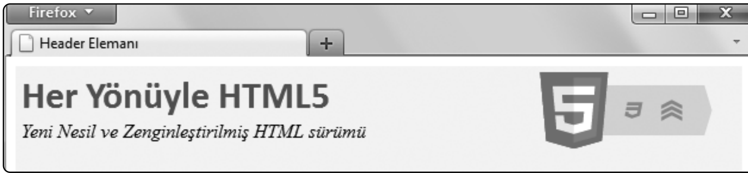
Özellikleri: `[Standart Özellikler]`

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Header Elemanı</title>
    <style type="text/css">
        header
        {
            width: 600px;
            height: 85px;
            background-color: #e8ff8c;
            margin: 0px auto;
            padding: 5px;
        }
        header h1
        {
            font-family: calibri;
            color: #ad3f23;
            margin: 0px;
        }
        header img
        {
            float: right;
        }
        header span
        {
            font-style: italic;
        }
    </style>
</head>
<body>
```

```
    <header>
        <img alt="html5" src="html_5.png"/>
        <h1>Her Yönüyle HTML5</h1>
        <span> Yeni Nesil ve Zenginleştirilmiş HTML sürümü</span
    </header>
</body>
</html>
```

Yukarıdaki uygulamada site için örnek bir başlık alanı oluşturulmuştur. Ekran görüntüsüne bakalım.



Firefox 4.0 ekran görüntüsü

<HGROUP>

Başlık elemanlarını (*heading elements*, `h1` .. `h6`) gruplamak ve bir belge ya da bölüm başlığı oluşturmak için kullanılır.

Özellikleri: `[Standart Özellikler]`

Örneğin;

```
<hgroup>
      <h1>Web Developers</h1>
      <h2>HTML5</h2>
</hgroup>
```

<NAV>

Navigasyon (yönlendirme, link) alanları oluşturmak için kullanılır. Burada şunu belirtelim <nav> elemanı link oluşturmak için kullandığımız <a> elemanının ya da `<ul>`, `<ol>` elemanlarının yerlerine kullanılmaz. Sadece bu elemanları kapsar (başka elemanları da kapsayabilir) ve içerdiği eleman gurubunu bir navigasyon alanı olarak tanımlar.

Özellikleri: `[Standart Özellikler]`

Örneğin;

```
<div id="nav_1">
    <ul>
        <li><a href="#">Birinci Link</a></li>
        <li><a href="#">İkinci Link</a></li>
        <li><a href="#">Üçüncü Link</a></li>
        <li><a href="#">Dördüncü Link</a></li>
    </ul>
</div>
```

...şeklindeki bir kullanım yerine aşağıdaki kullanımı tercih edebilirsiniz.

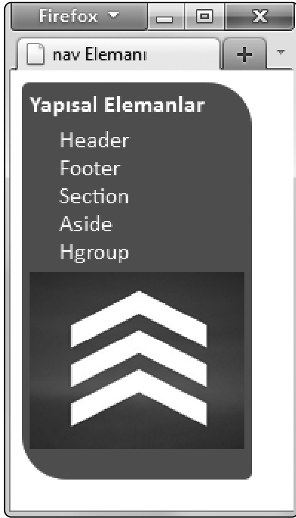
```
<nav>
    <ul>
        <li><a href="#">Birinci Link</a></li>
        <li><a href="#">İkinci Link</a></li>
        <li><a href="#">Üçüncü Link</a></li>
        <li><a href="#">Dördüncü Link</a></li>
    </ul>
</nav>
```

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>nav Elemanı</title>
    <style type="text/css">
        nav
        {
            width: 145px;
            height: 260px;
            background-color: #ae364c;
            padding: 5px;
            font-family: calibri;
            /*CSS3 border-radius özelliği yuvarlak kenarlı kutular oluşturmamızı sağlar.*/
            /* IE9,Firefox 4.0, Opera 10.5+, Safari 4.0,5.0 için [CSS3 Standardı]*/
            border-radius: 5px 30px;
            /*safari 3.0*/
            -webkit-border-radius: 5px 30px;
```

```
                /*Firefox 3.6 ve alt sürümleri için*/
                -moz-border-radius: 5px 30px;
            }
            nav h4
            {
                color: white;
                margin-left: 8px;
                margin: 0px;
            }
            nav > ul
            {
                margin: 5px 0px;
                padding-left: 20px;
                list-style-type: none;
            }
            nav > ul a
            {
                text-decoration: none;
                color: #ffff00;
            }
        </style>
    </head>
    <body>
        <nav>
                <h4>Yapısal Elemanlar</h4>
                <ul>
                    <li><a href="#">Header</a></li>
                    <li><a href="#">Footer</a></li>
                    <li><a href="#">Section</a></li>
                    <li><a href="#">Aside</a></li>
                    <li><a href="#">Hgroup</a></li>
                </ul>
                <img alt="res" src="semantics.png">
        </nav>
    </body>
    </html>
```

Ekran görüntüsüne bakalım.



Firefox 4.0 Beta11 ekran görüntüsü

<ARTICLE>

Sayfanın ana içerik alanında anlamsal bir bütünlüğü olan (farklı konu başlıklarına ya da davranışlara göre ayrılmış) alt bölümler (içerikler) oluşturmak için kullanılır. Örneğin; bir blog sayfasında ana içerik bölümü içindeki farklı blog yazı alanlarını bu elemanla oluşturabilirsiniz. Ya da gazete, dergi makaleleri içeren alanlar, kullanıcı girişi, kullanıcı yorumları gibi alanları bu elemanla oluşturmanız mümkündür. `article` elemanını kullanarak biçimlendirdiğiniz alanları sayfanın değişik bölgelerinde kullanabilirsiniz.

Özellikleri: `[Standart Özellikler]`

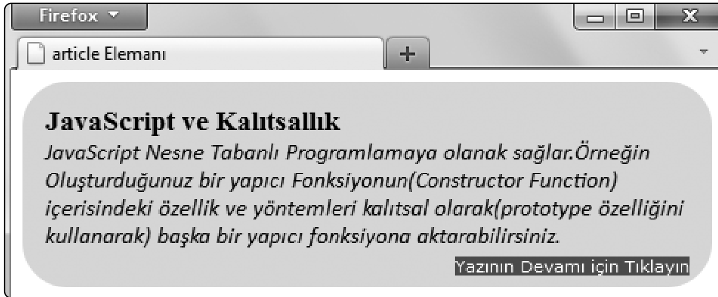
Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>article Elemanı</title>
    <style type="text/css">
        p,h3
        {
            margin: 0px;
        }
```

```
        article p
        {
            font-family: calibri;
            font-style: italic;
        }
        footer
        {
            text-align: right;
        }
        footer a
        {
            background-color: hsl(306, 83%, 36.9%); /*HSL hue (renk) -
            saturation (doygunluk) – lightness (parlaklık, aydınlık) hsl (H,S,L)
            CSS3 ile beraber gelen renk değeri tanımlama metodudur.
            hue değeri 0-360 arasında bir sayı alabilir. Örneğin; 240 mavi,
            360 kırmızı gibi.
            saturation ve lightness tanımlamaları yüzde olarak yapılır.
            [IE9 alt sürümleri bu tanımlama metodunu desteklemez]
            */
            color: white;
            font-family: verdana;
            font-size: 11px;
            text-decoration: none;
        }
        article
        {
            width: 437px;
            height: 110px;
            background-color: hsl(0, 100%, 91.4%);
            padding: 15px;
            /*IE9(Alt sürümleri desteklemiyor), Firefox 4.0, Opera 10.5+,
            Safari 4.0,5.0 için [CSS3 Standardı]
            */
            border-radius: 30px;
            /*safari 3.0 */
            -webkit-border-radius: 30px;
            /*Firefox 3.6 ve alt sürümleri için */
            -moz-border-radius: 30px;
        }
    </style>
```

```
</head>
<body>
    <article>
            <header>
                    <h3>JavaScript  ve Kalıtsallık</h3>
            </header>
           <p>JavaScript Nesne Tabanlı Programlamaya olanak sağlar.
             Örneğin Oluşturduğunuz bir yapıcı Fonksiyonun(Constructor
             Function) içerisindeki özellik ve yöntemleri kalıtsal
             olarak (prototype özelliğini kullanarak) başka bir yapıcı
             fonksiyona aktarabilirsiniz.
           </p>
            <footer>
                    <a href="#">Yazının Devamı için Tıklayın</a>
            </footer>
   </article>
</body>
</html>
```

Ekran görüntüsüne bakalım.



Firefox 4.0 ekran görüntüsü

<SECTION>

Sayfa içerisinde genel bölümler (ana içerik bölümü, diğer bölümler vb.) oluşturmak için kullanılır. HTML5 bu elemanı mantıksal, tematik alanlar (Örneğin; eğer şöyle derseniz birinci <section> elemanı içerisinde bulunan etiketler sayfanın ana `icerik` bölümünü oluşturuyor. Bu durumda `section` elemanının oluşturduğu alan mantıksal bir alan olarak ifade edilebilir) oluşturmak için tanımlasa da biz `section` elemanını sadece mantıksal alanlar oluşturmak için değil, aynı zamanda div elemanının yerine biçimlendirilmiş ana bölümler oluşturmak içinde kullanabiliriz.

Özellikleri: [Standart Özellikler]

Örnek:

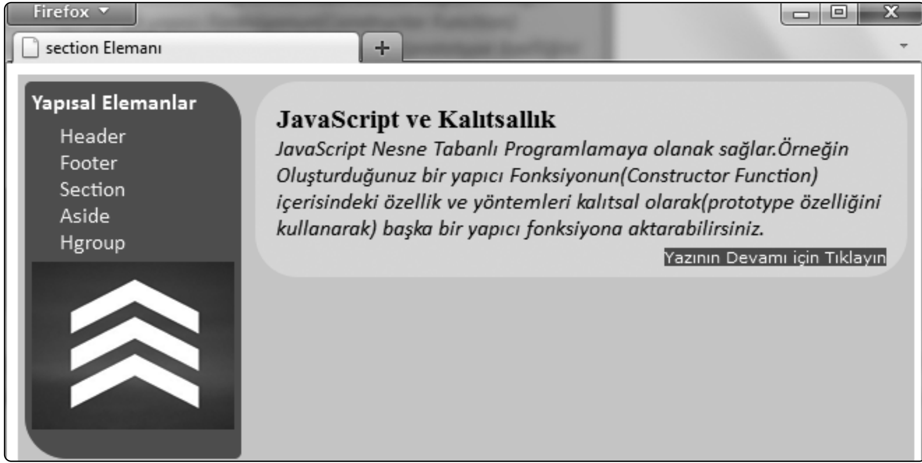
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>section Elemanı</title>
    <style type="text/css">
        div#content
        {
            margin: 0px auto;
            width: 642px;
            height: 280px;
            background-color:lightblue;
        }
        section
        {
            float: left;
            margin: 5px;
        }
        p, h3
        {
            margin: 0px;
        }
        article p
        {
            font-family: calibri;
            font-style: italic;
        }
        footer
        {
            text-align: right;
        }
        footer a
        {
            background-color: hsl(306, 83%, 36.9%);
            color: white;
            font-family: verdana;
            font-size: 11px;
            text-decoration: none;
```

```
}
article
{
     width: 437px;
     height: 110px;
     background-color: hsl(0, 100%, 91.4%);
     padding: 15px;
     /*IE9(Alt sürümleri desteklemiyor), Firefox 4.0, Opera 10.5+,
     Safari 4.0,5.0 için [CSS3 Standardı] */
     border-radius: 30px;
     /*safari 3.0 */
     -webkit-border-radius: 30px;
     /*Firefox 3.6 ve alt sürümleri için */
     -moz-border-radius: 30px;
}

nav
{
     width: 145px;
     height: 260px;
     background-color: #ae364c;
     padding: 5px;
     font-family: calibri;
     border-radius: 5px 30px;
     -webkit-border-radius: 5px 30px;
     -moz-border-radius: 5px 30px;
}
nav h4
{
     color: white;
     margin-left: 8px;
     margin: 0px;
}
nav > ul
{
     margin: 5px 0px;
     padding-left: 20px;
     list-style-type: none;
}
nav > ul a
{
```

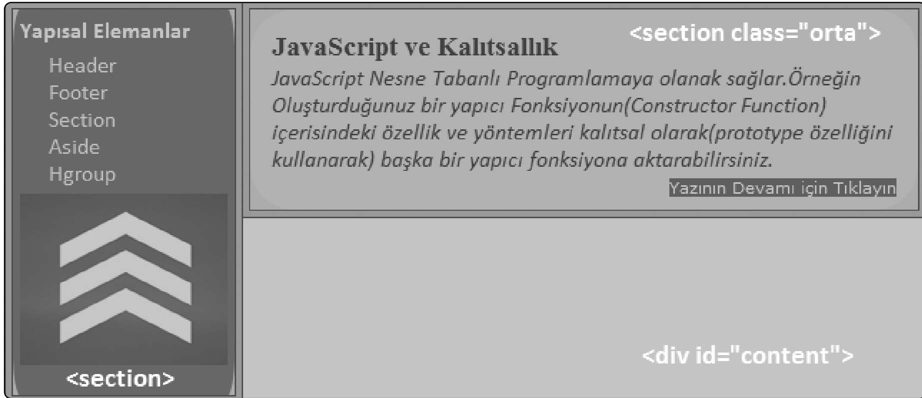
```
                text-decoration: none;
                color: #ffff00;
            }
        </style>
    </head>
    <body>
        <div id=”content”>
            <section>
                <nav>
                    <h4>Yapısal Elemanlar</h4>
                        <ul>
                            <li><a href=”#”>Header</a></li>
                            <li><a href=”#”>Footer</a></li>
                            <li><a href=”#”>Section</a></li>
                            <li><a href=”#”>Aside</a></li>
                            <li><a href=”#”>Hgroup</a></li>
                        </ul>
                         <img alt=”res” src=”semantics.png”>
                </nav>
            </section>
            <section class=”orta”>
                <article>
                     <header>
                          <h3>JavaScript ve Kalıtsallık</h3>
                    </header>
                    <p>JavaScript Nesne Tabanlı Programlamaya olanak
                       sağlar. Örneğin Oluşturduğunuz bir yapıcı
                       Fonksiyonun(Constructor Function) içerisindeki
                       özellik ve yöntemleri kalıtsal olarak(prototype
                       özelliğini kullanarak) başka bir yapıcı
                       fonksiyona aktarabilirsiniz.
                    </p>
                    <footer>
                          <a href=”#”>Yazının Devamı için Tıklayın</a>
                    </footer>
                </article>
            </section>
        </div>
    </body>
    </html>
```

Sayfamızın ekran görüntüsüne bakalım.



Firefox 4.0 ekran görüntüsü

Elemanların yerleşimine görsel olarak bakalım.



<ASIDE>

İçerik bölümleri ile alakalı ek bilgi alanları oluşturmak için kullanılırlar. Bu durumda içerik bölümünün yanına konumlandırılırlar. Ya da sayfanın ana bölümlerinin dışında sayfa kenarlarında reklam alanları, yan açıklama ve ek bilgi alanları tanımlamak için kullanılırlar.

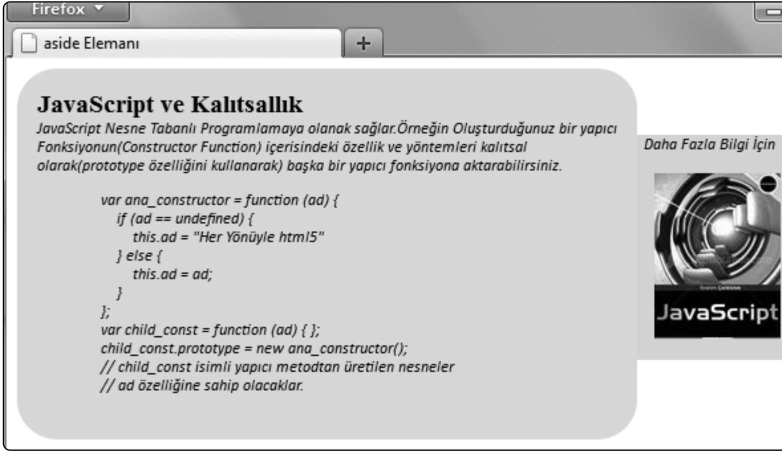
Özellikleri: [Standart Özellikler]

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>aside Elemanı</title>
    <style type="text/css">
        p,h3
        {
            margin: 0px;
        }
        article p, article pre, aside
        {
            font-family: calibri;
            font-style: italic;
            font-size: 12px;
        }
        article
        {
            width: 437px;
            height: 250px;
            background-color:#F9DAAE;
            padding: 15px;
            position: relative;
            border-radius: 30px;
            -webkit-border-radius: 30px;
            -moz-border-radius: 30px;
        }
        aside
        {
            position: absolute;
            top: 50px;
            right: -120px;
            background-color: #DEE17C;
            width: 120px;
            height: 170px;
            text-indent: 5px;
        }
```

```
    </style>
</head>
<body>
    <article>
           <header>
                    <h3>JavaScript ve Kalıtsallık</h3>
           </header>
           <p>JavaScript Nesne Tabanlı Programlamaya olanak sağlar.
             Örneğin Oluşturduğunuz bir yapıcı Fonksiyonun(Constructor
             Function) içerisindeki özellik ve yöntemleri kalıtsal
             olarak(prototype özelliğini kullanarak) başka bir
             yapıcı fonksiyona aktarabilirsiniz.
           </p>
           <pre>
                var ana_constructor = function (ad) {
                     if (ad == undefined) {
                          this.ad = "Her Yönüyle html5"
                     } else {
                          this.ad = ad;
                     }
                };
                var child_const = function (ad) { };
                child_const.prototype = new ana_constructor();
                // child_const isimli yapıcı metodtan üretilen nesneler
                // ad özelliğine sahip olacaklar.
           </pre>
           <aside>
             Daha Fazla Bilgi İçin
             <img src="js.png" alt="js"/>
           </aside>
   </article>
</body>
</html>
```

Ekran görüntüsüne bakalım:



Firefox 4.0 ekran görüntüsü

<FOOTER>

Bu elemanı kullanarak sayfa, bölüm ya da alt bölümler için alt bilgi alanları oluşturabilirsiniz. Alt bilgi alanları genelde içerik yazarı hakkında bilgi, içeriğin oluşturulma tarihi ya da sayfa sahibi ile ilgili iletişim bilgilerini içerir.

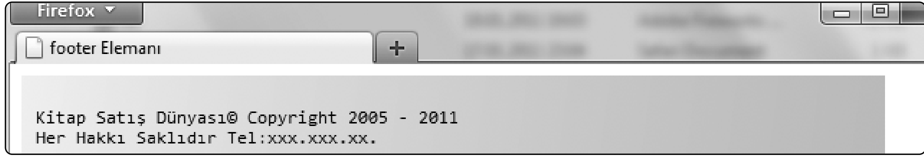
Özellikleri: [Standart Özellikler]

Örnek:

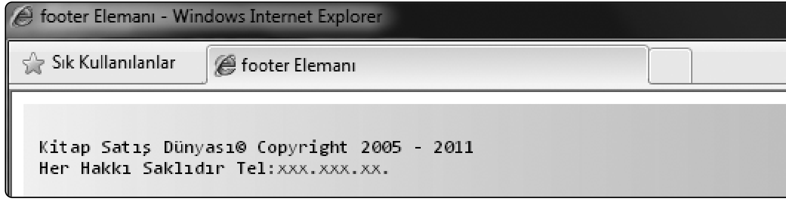
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8"/>
        <title>aside Elemanı</title>
        <style type="text/css">
            footer {
              /*IE9 ,firefox 4.0 ,Safari 5.0 alt sürümleri için
               display:block tanımlaması yapılmıştır*/
                display:block;
                width:600px;
                height:70px;
                font-family:consolas;
                font-size:12px;
                padding-top:10px;
```

```
                    /*Arkaplan rengi olarak bir gradient tanımlama (renk geçişi)*/
                    /*Mozilla Firefox 3.6+*/
                    background:-moz-linear-gradient(left top,#F5F3F0,#2CDEDE);
                    /*Safari 4.0+*/
                    background:-webkit-gradient(linear, 46% 0%, 100% 100%,
from(#F5F3F0), to(#2CDEDE));
                    /*IE 5.5+*/
filter:Progid:DXImageTransform.Microsoft.gradient(startColorstr='#F5F3F0',
endColorstr='#2CDEDE',gradientType="1");
                    }
               footer p {
                    width:300px;
                    font-family:consolas;
                    font-size:12px;
                    padding-left:10px;
               }
          </style>
          <!--[if lt IE 9]>
          <script>
          document.createElement("footer");
          </script>
          <![endif]-->
     </head>
     <body>
          <footer>
               <p>
                    Kitap Satış Dünyası© Copyright 2005 - 2011
                    Her Hakkı Saklıdır Tel:xxx.xxx.xx.
               </p>
          </footer>
     </body>
</html>
```

Ekran görüntülerimize bakalım:



Firefox 4.0 ekran görüntüsü



IE8 ekran görüntüsü

<FIGURE>

Resim, fotoğraf alanları (ya da media alanları) tanımlamak için kullanılır. Bu şekil alanlarına bir metin iliştirmek için <figcaption> elemanı kullanılır.

Özellikleri: [Standart Özellikler]

Örneğin;

```
<figure>
      <img src="arge.jpg" width="304" height="228" />
</figure>
```

ya da resme tanımlayıcı bir metin ekleyebilirsiniz.

```
<figure>
      <img src="arge.jpg" width="304" height="228" />
      <figcaption>
          Şirketimizin Arge Bölümü
      </figcaption>
</figure>
```

NOT

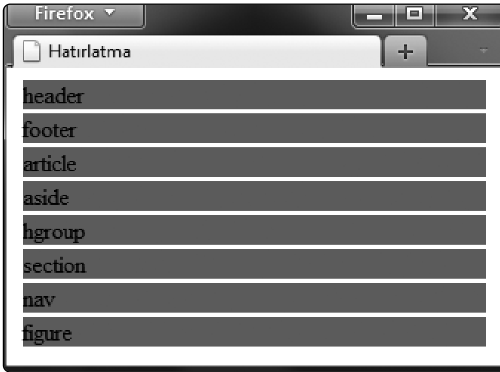
HATIRLATMA

Yukarıda anlatılan elemanların blok seviyesinden elemanlar olduklarını fark etmişsinizdir. HTML elemanları sayfadaki yerleşim düzeylerine göre ikiye ayrılır. Bunlar; **satır içi** (*inline element*), **blok seviyesi elemanı** (*block level element*)'dır. Tarayıcı blok seviyesi elemanından önce ve sonra bir satır sonu oluşturur.

Örnek:

```
<!DOCTYPE html>
<html>
    <head><meta charset=”utf-8”/>
        <title>Hatırlatma</title>
        <style type=”text/css”>
            body> * {
                background-color: crimson;
            }
        </style>
    </head>
    <body>
        <header>
            header
        </header>
        <footer>
            footer
        </footer>
        <article>
            article
        </article>
        <aside>
            aside
        </aside>
        <hgroup>
            hgroup
        </hgroup>
        <section>
            section
        </section>
        <nav>
            nav
        </nav>
        <figure>
            figure
        </figure>
    </body>
</html>
```

Ekran görüntüsü:



Firefox 4.0Beta11 ekran görüntüsü

Yukarıda anlatılan elemanlar IE9, Firefox 4, Opera (Presto 2.7.70), Safari 5.0 tarayıcılarının alt sürümleri tarafından desteklenmemektedir. Peki, ya bu alt sürümler bu elemanı sayfaya nasıl yerleştirecekler? Hemen açıklayalım.

Internet Explorer alt sürümleri bu elemanları hiç tanımayacaktır. Bu elemanları hedef alan CSS kodlarını uygulamazlar. Bu durumun çözümü için JavaScript yardımıyla belirtilen elemanları programatik olarak oluşturmak yeterli olacaktır. IE9'un tanımadığı diğer yeni elemanları da bu şekilde oluşturup sayfa içerisinde kullanabilirsiniz.

```
        <!--[if lt IE 9]>
        <script type="text/javascript">
        var elemanlar=["header","hgroup","nav","article","section","aside",
"footer","figure"];
        for(var i=0;i<elemanlar.length;i++){
        document.createElement(elemanlar[i]);
        /*Yeni bir element (eleman) oluşturmak için kullanılır.
              *Kullanımı:
                *createElement(elementName):HTMLElement
                *Örneğin; document.createElement("div");
        *DOM Level 1,2
        */
        }
        </script>
        <![endif]-->
```

Yukarıdaki Script bloğu sadece IE9 alt sürümlerinde çalışacaktır.

Belirtilen elemanları IE9 alt sürümlerinde blok seviyesin den bir eleman olarak kullanmak için aşağıdaki CSS tanımlamasını yapalım.

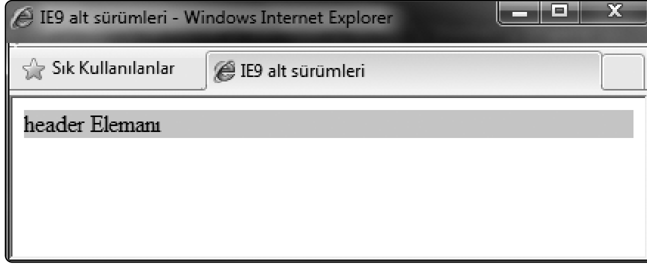
```
header, hgroup, nav, article, section, aside, footer, figure {
       display:block;
}
```

Artık IE9 alt sürümlerinde bu elemanları kullanabiliriz.

Örnek:

```
<!DOCTYPE html>
<html>
    <head><meta charset="utf-8"/>
        <title>IE9 alt sürümleri</title>
        <style type="text/css">
            header, hgroup, nav, article, section, aside, footer, figure {
                display:block;
            }
            header {
                background-color:lightblue;
            }
        </style>
        <!--[if lt IE 9]>
        <script type="text/javascript">
        var elemanlar=["header","hgroup","nav","article","section",
"aside","footer","figure"];
        for(var i=0;i<elemanlar.length;i++){
           document.createElement(elemanlar[i]);
        }
        </script>
        <![endif]-->
    </head>
    <body>
        <header>
            header Elemanı
        </header>
    </body>
</html>
```

Ekran görüntüsüne bakalım:



IE8 ekran görüntüsü

Firefox 4.0 ve **Opera** (*Presto 2.7.70*) gibi diğer tarayıcıların alt sürümleri bu elemanları satır içi bir eleman olarak oluştururlar oysa ki bu elemanlar sayfa bölümleri ve alt bölümler oluşturmak için kullanılırlar (Blok seviyesinden elemanlar olmaları gerekir). Bu problemin çözümü için aşağıdaki CSS tanımlaması yeterli olacaktır.

```
header, hgroup, nav, article, section, aside, footer, figure {
    display:block;
}
```

Diğer Elemanlar

Yukarıda anlatılan yapısal elemanlar dışında HTML5 yapısı içerisinde duyurulan diğer işlevsel elemanlara bakalım.

<mark>

Metni işaretlemek için kullanılır. Bu elemanı kullanarak okuyucunun dikkatini çekecek vurgulu metinler oluşturabilir ya da metnin referans olarak tanımlanmasını sağlayabilirsiniz. Satır içi bir elemandır.

Özellikleri: `[Standart Özellikler]`

Örneğin;

```
<p>
       Metni
       <mark>
              işaretlemek
       </mark>için kullanılır.
</p>
```

CSS kodları ile mark elemanı içerisindeki metne stil vererek okuyucunun dikkatini bu kelimeye çekebilirsiniz.

Örnek:

```
<p>
        Bu durumun çözümü için JavaScript yardımıyla belirtilen
        elemanları programatik olarak oluşturmamız yeterli olacaktır.
        <mark> IE9</mark>'un tanımadığı diğer yeni elemanları da bu
        şekilde oluşturup <mark> IE9</mark> alt sürümlerinde
        ortaya çıkan bu problemi çözebilirsiniz.
</p>
```

Sayfada yapılacak aramalarda bulunacak aynı metin parçalarının hepsini bu etiket içerisine alıp, önceden bir stil tanımlaması yapabilirsiniz.

<METER>

Belirlediğiniz bir aralık içinde bir değeri (ölçümü) kullanıcıya görsel olarak göstermek için kullanılır. Bu eleman bir ilerleme çubuğu (*progress bar*) olarak kullanılmamalıdır.

Özellikleri: `[Standart Özellikler]` ile `form`, `high`, `low`, `max`, `min`, `optimum`, `value`

- **`form:`** `meter` elemanının bir form elemanı ile ilişkili olup olmadığını tanımlamak için kullanılır. Bu özelliğe elemanın ilişkili olduğu form elemanının `id` özelliğine atanan değer yazılır. Aralarında boşluk bırakmak kaydıyla birden fazla form `id` değeri yazılabilir.
- **`high:`** `min`, `max` özellikleri ile tanımlanan aralıkta yüksek olarak kabul edilecek değeri tanımlar. Belirtilmezse ya da `high` özelliğine atanan değer `max` özelliğine atanan değerden yüksek ise `max` özelliğine atanan değer yüksek değer olarak kabul edilir. `low` ve `optimal` özellikleri ile beraber kullanılır.
- **`low:`** `min`, `max` özellikleri ile tanımlanan aralıkta düşük olarak kabul edilecek değeri tanımlar. Belirtilmezse ya da `low` özelliğine atanan değer `min` özelliğine atanan değerden düşükse, `min` özelliğine atanan değer, düşük değer olarak kabul edilir. `high` ve `optimal` özellikleri ile beraber kullanılır.
- **`max, min:`** Aralığın en düşük ve en yüksek değerlerini tanımlamak için kullanılırlar. Ölçüm, (sonuçta kullanıcıya görsel olarak sunulacak değer) bu değerler arasındadır. Eğer bu özellikler tanımlanmazsa varsayılan olarak `max=1.0`, `min=0` değerlerini alır.

- **optimum:** min, max değerleri arasında en uygun değer tanımlaması yapmak için kullanılır. Bu değer high değerinden büyük ise yüksek değerler, low değerinden küçük ise küçük değerler uygun değerler olarak kabul edilebilir. Belirtilmezse min, max değerleri arasındaki orta nokta optimum değer olarak kabul edilir.
- **value (Gerekli):** Ölçüm değerini (sonuçta kullanıcıya görsel olarak sunulacak değer) tanımlamak için kullanılır. Bu değer min, max değerleri arasında olmalıdır. Bu özelliğe değer atanmazsa varsayılan değer 0 olur. min değerden daha düşük olursa value=min, max değerden daha büyük olursa value=max olur.

Aşağıdaki örnekte value değeri low (aralıkta düşük olarak kabul edilen değer) değerinden daha küçüktür.

```
<p>
Ölçülen Açı:
<meter min="0" max="180" low="30" optimum="90" high="120"
value="25"></meter>
</p>
```

Ekran görüntüsü:



Opera 11.ekran görüntüsü

Şimdi de value değerine high (aralıkta yüksek olarak kabul edilen değer) değerinden daha büyük bir değer verelim.

```
<p>
Ölçülen Açı:
<meter min="0" max="180" low="30" optimum="90" high="120"
value="130"></meter>
</p>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

Dikkat ederseniz her iki durumda da ilerleme çubuğunun dolgu rengi aynı olacaktır. Çünkü değerlerden birincisi düşük olarak kabul edilen değerden (low) daha küçük diğeri ise büyük olarak kabul edilen değerden (high) daha büyüktür.

value özelliğine low, high değerlerinin arasında bir değer atanırsa, ilerleme çubuğunun dolgu rengi yukarıdaki durumlardan farklı olacaktır (yeşil olur).

```
<p>
Ölçülen Açı:
<meter min="0" max="180" low="30" optimum="90" high="120"
value="115"></meter>
</p>
```

Ekran görüntüsü:



Opera 11.01 görüntüsü

<COMMAND>

Sayfada görüntülenen (Örneğin; menu elemanı içerisinde) ya da görüntülenmeyen bir komut düğmesi tanımlamak için kullanılır. Type özelliğini kullanarak bu elemanı checkbox ya da radio elemanlarına dönüştürebilirsiniz. Bu elemana şu an için tarayıcı desteği bulunmamaktadır.

Özellikleri: [Standart Özellikler] ve checked, disabled, icon, label, radiogroup, type

<PROGRESS>

Bir görevin, işlemin tamamlanma/ilerleme sürecini kullanıcıya göstermek için yani ilerleme çubuğu oluşturmak için kullanılır.

Özellikleri: [Standart Özellikler] ve max,value

- **max:** Görevin ya da işlemin bitirilme durumunu tanımlayan değer.
- **value:** Görevin ya da işlemin şu andaki durumunu gösteren değer.

Örnek:

```
<p>
        İşlemin Durumu:
        <progress value="78" max="100">
        </progress>
</p>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

<TIME>

Zaman ya da tarih ya da her ikisini birden içeren tanımlamalar yapmak için kullanılır.

Özellikleri: `[Standart Özellikler]` ve `datetime, pubdate`

`datetime` özelliği tarih ya da zaman tanımlamak için kullanılır. Aşağıdaki formata göre tarih ya da zaman değerlerini girebilirsiniz.

`Yıl-Ay-Gün Saat:Dakika:Saniye TZD (Time Zone Designator, Saat dilimi Tanımlayıcısı)`

Yukarıdaki tanımlamada kullanılan argümanlar isteğe bağlıdır.

Örnek:

```
<p>İşe Başlama Tarihi: <time datetime="2010-01-10 9:00">Ocak 10</time>.</p>
```

HTML5 VE JAVASCRIPT

3

HTML5 yapısı kendinden önceki HTML sürümleri gibi JavaScript programlama dilini kullanır. JavaScript, tarayıcı da çalışan ve Nesne Tabanlı Programlama imkanı sunan bir dildir. Browser, JavaScript komutlarını yorumlayarak işlevlerini yerine getirir. JavaScript **ECMAScript-262** (5. sürüm) standardını kullanır. Bilinmesi gereken diğer bir konuda **DOM** (**D***ocument* **O***bject* **M***odel*)'dir. DOM, HTML ve XML belgeleri için oluşturulmuş bir programlama arayüzüdür (API).

DOM (*Doküman Nesne Modeli*), sayfadaki tüm elemanları birer nesne olarak tanımlar. JavaScript yardımıyla DOM'un tanımladığı bu nesnelere (nesnelerin özellik ya da metotlarına) ulaşabilir ve kod yazabiliriz. DOM bildirimleri sayfa elemanlarının dışında sayfanın işleyişi, olay akışı ve diğer durumlar için özel nesneler tanımlar ve kullanıma sunar.

Aşağıda DOM bildirimleri verilmiştir.

Document Object Model Level 1

- Document Object Model Level 1 (W3C Recommendation)
- Document Object Model Level 1 (Second Edition) (W3C Working Draft)

Document Object Model Level 2

- Document Object Model Level 2 Core (W3C Recommendation)
- Document Object Model Level 2 Views (W3C Recommendation)
- Document Object Model Level 2 Events (W3C Recommendation)

- Document Object Model Level 2 Style (W3C Recommendation)
- Document Object Model Level 2 Traversal and Range (W3C Recommendation)
- Document Object Model Level 2 HTML (W3C Recommendation)

Document Object Model Level 3

- Document Object Model Level 3 Core (W3C Recommendation)
- Document Object Model Level 3 Load and Save (W3C Recommendation)
- Document Object Model Level 3 Validation (W3C Recommendation)

Diğer Bildirimler (Sadece Selector Api'ler listelenmiştir)

- Selectors API Level 1
- Selectors API Level 2 (W3C Working Draft)

Dikkat ederseniz; DOM bildirimleri parça parça yapılmış olup, **core** bildirimleri ilgili sürümün çekirdek yapısını oluşturmaktadır. Tarayıcılar açısından olaya bakarsak; modern web tarayıcılarının yeni modelleri DOM Level 2 bildirimlerine büyük bir ölçüde destek vermekle beraber, DOM Level 3 bildirimlerine desteklerini hızla açıklamaktadırlar. DOM bildirimlerinde tanımlanan nesnelere, nesnelerin özellik ya da metotlarına tarayıcıların eski sürümleri ya kısmen destek verirler ya da hiç vermezler. Bu durum bizi farklı tarayıcılar için farklı kodlar yazmaya yöneltir.

Örneğin; Document Object Model Level 2 Events bildirimi ile tanımlanan olay akışı üç evreden oluşur. Bunlar; **capturing**, **target**, **bubling**'dir. Peki, bu modeli tarayıcılar destekliyor mu? Yani Document Object Model Level 2 Events bildirimine tarayıcı desteği nedir?

IE9 alt sürümleri olay akışını sadece bubling evresinden ibaret görür. Yani diğer evreleri desteklemez ve bu bildirim içinde bulunan olay dinleyicilerini de desteklemediğinden, biz de uygulama oluştururken farklı tarayıcı kodları yazarız. Söylemek istediğim şey; tarayıcıların bu bildirimleri farklı oranda destekledikleridir. Peki, HTML5 ile bunun ne alakası var?

HTML5 tarayıcıların DOM bildirimlerine verdikleri desteği hızlandırmalarına neden olmuştur. Tarayıcılar HTML5 verdikleri destekleri açıkladıkları sayfalarda DOM, JavaScript ve diğer ek teknolojilere verdikleri destekleri de açıklamaktadırlar. HTML5, JavaScript, DOM iç içe geçmiş yapılardır.

Tarayıcılar, DOM nesnelerini ve nesneler için tanımlanan özellik ve metotları destekledikçe istemci taraflı web uygulamaları geliştirmek daha konforlu bir hal alacaktır. JavaScript dilinin etkinliği (gücü) artacak ve HTML5 dilini daha etkili bir şekilde kullanabileceğiz. Şimdi aşağıda HTML5 yapısı ile beraber tarayıcıların destek verdiklerini açıkladıkları ve **W3C Selectors API Level 1** bildiriminde bulunan yeni metotlara bakalım.



Tarayıcıların DOM ve CSS bildirimlerine verdikleri destekleri görmek için tarayıcı destek sayfalarına ya da *http://www.quirksmode.org/* sitesine bakabilirsiniz.

querySelector()

W3C Selectors API Level 1 bildiriminde bulunan bu metot ile CSS seçicilerini kullanarak belgedeki bir elemanın referansını alabilirsiniz.

Kullanımı: `elemanReferansı=document.querySelector(selectors)`

İlk durumda; belge içerisinde seçici tarafından hedef alınan ilk elemanın referansını geriye döndürür.

```
elemanReferansı=temelEleman.querySelector(selectors)
```

İkinci durumda; belirtilen temel eleman içinde seçici tarafından hedef alınan ilk elemanın referansını geriye döndürür.

Eğer seçici tarafından hedef alınan eleman bulunamazsa (seçici ile eşleşen eleman olmazsa) geriye `null` değeri döner.

Seçici tarafından hedef alınmak ne demektir?

CSS seçicileri, HTML eleman isimleri ve birçok özel karakterler içerebilir. Oluşturduğunuz seçiciler en kaba tabiriyle içinde barındırdıkları CSS kodlarının uygulanması için eleman ya da elemanlar seçer (bu açıdan seçiciler birer sorgu deyimi gibidirler). Seçici tanımlamaları yaparak CSS kodlarının uygulanacağı elemanlar seçilir, yani hedef alınır.

Örnek:

```
<nav>
	<ol>
		<li>Yeşil
		</li>
```

```
                <li id="acil">Beyaz
                </li>
                <li>Kırmızı
                </li>
                <li>Sarı
                </li>
                <li>Mavi
                </li>
        </ol>
</nav>
```

Seçicilerimize bakalım...

```
nav ol li {
        /*Bu seçici sırasıyla nav ve ol elemanlarının soyundan
        gelen li elemanlarını hedef alır (seçer).*/
        color:blue;
}
nav ol> li#acil+ li {
        /*Bu seçici sırasıyla nav ve ol elemanlarının soyundan
        gelen li#acil elemanının kardeşi olan li elemanını hedef alır (seçer).*/
        color:red;
}
```

Ekran görüntüsü:



Firefox 4.0 Beta11 ekran görüntüsü

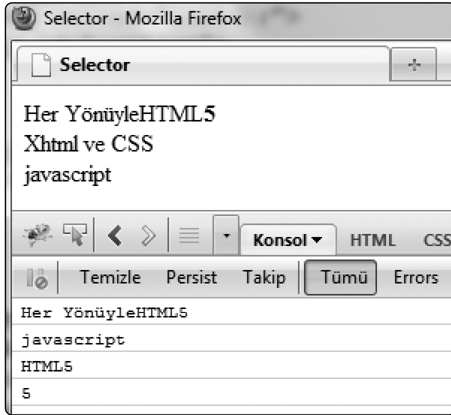
Örnek:

```
<!DOCTYPE html>
<html>
```

```
        <head><meta charset="utf-8">
        <title>Selector</title>
        <style type="text/css">
        </style>
        <script type="text/javascript">
        /*
        * Direk script etiketleri arasında(sayfa yüklenmeden önce) bir
        * elemanın referansını alamayız.
        * çünkü tarayıcı bu satırları body elemanını yorumlamadan (sayfaya
        * yüklemeden) önce okuyacağından belirtilen elemanı bulamaz ve
        * "null" değerini döndürür.
        */
        var init= function() {
                /* ilk önce id özelliğine html5 değeri atanmış elemana
                ulaşalım (referansını alalım, nesne olarak elde edelim)*/
                var elemanHtml5=document.querySelector("#html5");
                /*element.textContent:String (geriye String değer döndürür)
                Düğüm içerisindeki metni verir.
                Bildirim: DOM Level 3 Core */
                console.log(elemanHtml5.textContent);
                /* div#content elemanı içerisindeki en son div elemanına ulaşalım.*/
                var elemanJs=document.querySelector("div#content>div#css+div");
                console.log(elemanJs.textContent);
                /* div#html5 elemanı içerisindeki span elemanının referansını alalım.*/
                var elemanSpan=elemanHtml5.querySelector("span");
                console.log(elemanSpan.textContent);
                /* Sırasıyla div#html5 ve span elemanlarının içinde olan b elemanının
                referansını alalım.*/
                var elemanB=elemanHtml5.querySelector("span>b");
                console.log(elemanB.textContent);
        }
                </script>
        </head>
        <body onload="init();">
        <div id="content">
                <div id="html5">Her Yönüyle<span>HTML<b>5</b></span></div>
                <div id="css">Xhtml ve CSS</div>
                <div>javascript</div>
        </div>
        </body>
</html>
```

Firebug; Firefox tarafından desteklenen, hata ayıklama için kullanabileceğimiz, CSS özelliklerinin eleman üzerindeki etkisini görsel olarak görebileceğimiz, elemanların DOM özellik ve yöntemlerini ayrıntılı bir şekilde gösteren ve diğer birçok özelliği barındıran web sayfası tasarımcıları için bulunmaz bir araçtır. Çalışma anında firebug konsoluna değişken, nesne ya da bir değer yazdırmak için `console.log()` metodu kullanılır.

Şimdi ekran görüntüsüne bakalım.



Firefox 3.6 ve Firebug 1.6.2 ekran görüntüsü

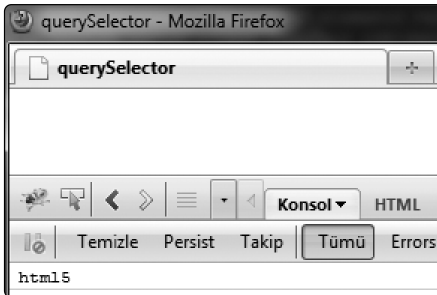
> **NOT** Firebug aracını *http://getfirebug.com/* sitesinden indirebilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>querySelector</title>
              <style type="text/css">
              </style>
              <script type="text/javascript">
              var init= function() {
              var elemanDiv=document.querySelector("div#content div")
              /*Oluşturduğumuz seçici ile aslında div#content elemanı içindeki
                 tüm div elemanları seçilmiş yani hedef alınmıştır.
                 Fakat querySelector() metodu seçici tarafından hedef alınan belge
                 ağacındaki ilk elemanın referansını döndürür.*/
```

```
                        console.log(elemanDiv.id);
                }
                </script>
        </head>
        <body onload="init();">
                <div id="content">
                        <div id="html5">
                        </div>
                        <div id="css">
                        </div>
                        <div id="js">
                        </div>
                </div>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ve Firebug 1.6.2 ekran görüntüsü

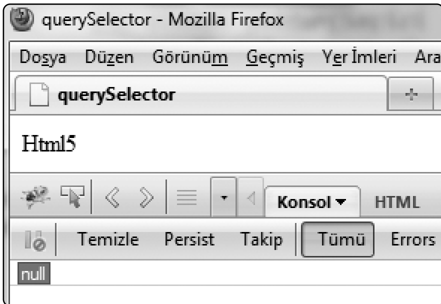
Elemanların yapısal olarak görüntüsü:



Örnek:

```
<!DOCTYPE html>
<html>
	<head><meta charset="utf-8">
		<title>querySelector</title>
		<style type="text/css">
		</style>
		<script type="text/javascript">
	var init= function() {
		var elemanDiv=document.querySelector("div#content>div")
		/*Oluşturduğumuz seçici ile div#content elemanının çocuğu olan
		ilk div elemanına ulaşmak istedik. Fakat böyle bir eleman yoktur (seçici ile
		eşleşen eleman yok).
		Dolayısıyla geriye "null" değeri döner.*/
		console.log(elemanDiv);
	}
		</script>
	</head>
	<body onload="init();">
		<div id="content">
			<span>Html5</span>
		</div>
	</body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ve Firebug 1.6.2 ekran görüntüsü

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>querySelector</title>
              <style type="text/css">
              </style>
              <script type="text/javascript">
var init= function() {
/*querySelector() metoduna arada virgül koymak kaydıyla bir ya da daha fazla seçici yazılabilir.
Bu durumda; eğer belge üzerinde birinci seçicinin hedef aldığı eleman bulunamazsa ikinci
seçicinin hedef aldığı eleman aranır.*/
var elemanDiv=document.querySelector("#div1,#div2");
var elemanP=document.querySelector(".p1,.p2");
alert(elemanDiv.id + "\n" + elemanP.className);
}
              </script>
       </head>
       <body onload=init();>
              <div id="div1">
                     <p>
                            Aptana Studio 3
                     </p>
                     <p class="p2">
                            Dom Level 2
                     </p>
              </div>
              <div id="div2">
              </div>
       </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

Tarayıcı desteği: Internet Explorer 8+, Firefox 3.5+, Opera 10+, Safari 3.2+

querySelectorAll()

W3C Selectors API Level 1 bildirimi ile tanımlanan bu metot CSS seçicilerini kullanarak belgedeki eleman ya da elemanları nesne olarak elde etmek için kullanılır.

Kullanımı: `elemanlistesi=document.querySelectorAll(selectors)`

İlk durumda; belge içerisinde seçici ile eşleşen elemanların listesini (`StaticNodeList`) döndürür.

`elemanlistesi=temelEleman.querySelector(selectors)`

İkinci durumda; Belirtilen temel eleman içinde seçici ile eşleşen elemanların listesini (`StaticNodeList`) döndürür.

Eğer seçici tarafından hedef alınan eleman ya da elemanlar bulunamazsa (seçici ile eşleşen eleman olmazsa) geriye `null` değeri döner.

`StaticNodeList` (*static düğüm listesi*); `querySelectorAll()` metodu ile `StaticNodeList` nesnesi elde edildikten sonra, DOM metotları ile belge ağaç yapısında programatik olarak yapılan değişiklikler olduğunda (eleman ekleme, çıkarma) `StaticNodeList` içindeki eleman listesi güncellenmez.

`StaticNodeList` nesnesinin (`querySelectorAll()` metodunu atadığımız değişken)

içindeki eleman referanslarına ulaşmak için; `StaticNodeList[index numarası]` ya da `StaticNodeList.item(index numarası)` yöntemlerinden birini kullanabilirsiniz. `index` numarası `0`'dan başlayacaktır.

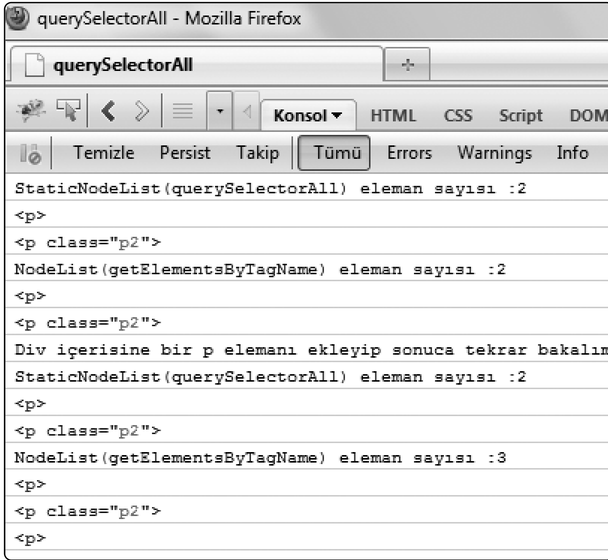
İlk örneğimizi `NodeList` ve `StaticNodeList` arasındaki farkı anlatmak için yapalım;

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset="utf-8">
                <title>querySelectorAll</title>
                <style type="text/css">
                </style>
                <script type="text/javascript">
        var init= function() {
                var elemanlar1=document.querySelectorAll("div#div1>*");
                /*div#div1 elemanı içerisindeki tüm elemanların static bir
                listesi(StaticNodeList) alındı(elemanlar1 içinde)*/
                var elemanlar2=document.getElementsByTagName("p");
                /*Belge içerisindeki tüm p elemanlarının canlı bir listesi
                  (NodeList) alındı (elemanlar2 içinde)*/
                /*StaticNodeList içindeki eleman sayısını yazdıralım*/
                console.log("StaticNodeList(querySelectorAll) eleman sayısı"+"
                :"+elemanlar1.length);
                /*StaticNodeList içindeki elemanları konsola yazdıralım*/
                for(var i=0;i<elemanlar1.length;i++) {
                        console.log(elemanlar1.item(i));
                }
                /*NodeList içindeki eleman sayısını yazdıralım*/
                console.log("NodeList(getElementsByTagName) eleman sayısı"+"
                :"+elemanlar2.length);
                /*NodeList içindeki elemanları konsola yazdıralım*/
                for(var i=0;i<elemanlar2.length;i++) {
                        console.log(elemanlar2.item(i));
                }
                console.log("Div içerisine bir p elemanı ekleyip sonuca
                tekrar bakalım");
                /*Yeni bir p elemanı oluşturulup bir text içeriği eklendikten
                sonra div#div1 içerisine ekleyelim*/
                var yeni_eleman=document.createElement("p");
```

```
                yeni_eleman.textContent="yeni bir div elemanı";
                var elemanDiv=document.querySelector("#div1");
                elemanDiv.appendChild(yeni_eleman);
                /*StaticNodeList içindeki eleman sayısını tekrar yazdıralım*/
                console.log("StaticNodeList(querySelectorAll) eleman sayısı"+"
                :"+elemanlar1.length);
                /*StaticNodeList içindeki elemanları konsola yazdıralım*/
                for(var i=0;i<elemanlar1.length;i++) {
                        console.log(elemanlar1.item(i));
                }
                /*NodeList içindeki eleman sayısını tekrar yazdıralım*/
                console.log("NodeList(getElementsByTagName) eleman sayısı"+"
                :"+elemanlar2.length);
                /*NodeList içindeki elemanları konsola yazdıralım*/
                for(var i=0;i<elemanlar2.length;i++) {
                        console.log(elemanlar2.item(i));
                }
        }
                </script>
        </head>
        <body onload=init();>
                <div id="div1">
                        <p>
                                Aptana Studio 3
                        </p>
                        <p class="p2">
                                Dom Level 2
                        </p>
                </div>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ve Firebug 1.6.2 ekran görüntüsü

Örnekten de anlaşılacağı üzere `NodeList`, canlı (güncellenen) bir eleman düğümü listesi iken, `StaticNodeList` güncellenmeyen statik düğüm listesidir.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8">
            <title>querySelectorAll</title>
            <style type="text/css">
            </style>
            <script type="text/javascript">
      var init= function() {
            var staticlistDiv=document.querySelector("div#div1>div");
            /*div#div1 elemanının çocuğu olan herhangi bir div elemanı
            olmadığından geriye null değeri döner*/
            console.log(staticlistDiv);
            // sonuç: "null"
      }
            </script>
      </head>
```

```
        <body onload=init();>
                <div id="div1">
                        <p>
                                Aptana Studio 3
                        </p>
                        <p class="p2">
                                Dom Level 2
                        </p>
                </div>
        </body>
</html>
```

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset="utf-8">
                <title>querySelectorAll</title>
                <style type="text/css">
                </style>
                <script type="text/javascript">
        var init= function() {
                var staticlistP=document.querySelectorAll("div#div1>p");
                /*div#div1 elemanının çocuğu olan p elemanlarının
                  listesi geriye döner*/
                for(var i=0;i<staticlistP.length;i++) {
                  /*NodeList ya da StaticNodeList nesnelerinin sakladığı eleman
                  sayısı length özelliği ile bulunur.*/
                  console.log(staticlistP[i].id);
                  // ya da  console.log(staticlistP.item(i))
                  /*Bir StaticNodeList nesnesinin(staticlistP)
                içindeki eleman referanslarına ulaşmak için;
                NodeList[index numarası] ya da NodeList.item(index numarası)
                yöntemlerinden birini kullanabilirsiniz.*/
                }
        }
                </script>
        </head>
        <body onload=init();>
                <div id="div1">
                        <p id="x">
```

```
                        Aptana Studio 3
                </p>
                <p id=”y”>
                        Dom Level 2
                </p>
                 <p id=”z”>
                        Dom Level 3
                </p>
        </div>
    </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ve Firebug 1.6.2 ekran görüntüsü

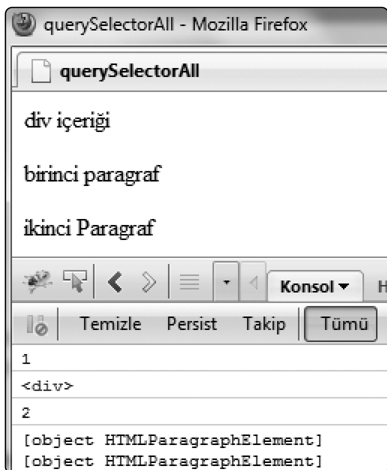
Örnek:

```
<!DOCTYPE html>
<html>
    <head><meta charset=”utf-8”>
        <title>querySelectorAll</title>
        <style type=”text/css”>
        </style>
        <script type=”text/javascript”>
    var init= function() {
        var staticlistDiv=document.querySelectorAll(“body>div”);
        /*belge içerisinde seçici ile eşleşen(seçici tarafından seçilen) sadece
        bir div elemanı var. Bu durumda StaticNodeList(StaticElemanListesi)
        içerisinde sadece bir div elemanı olacaktır*/
```

```
            console.log(staticlistDiv.length);
            console.log(staticlistDiv[0]);
            var staticlistP=document.querySelectorAll("p");
            /*
            belge içerisindeki tüm p elemanlarının listesi staticlistP nesnesine aktarıldı.
            Aslında  liste  türünü(NodeList,StaticNodeList) göz ardı ederseniz
            getElementsByTagName() metodu ile aynı işlemi yaptı*/
            console.log(staticlistP.length);
            console.log(staticlistP[0]+"\n" +staticlistP[1]);
        }
            </script>
        </head>
        <body onload=init();>
            <div>
                div içeriği
            </div>
            <p>
                birinci paragraf
            </p>
            <p >
                ikinci Paragraf

            </p>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ve
Firebug 1.6.2 ekran görüntüsü

Tarayıcı desteği: Internet Explorer 8+, Firefox 3.5+, Opera 10+, Safari 3.2 +

getElementsByClassName()

HTML5 bildirimi içinde bulunur. Bu metot, belirtilen sınıf seçiciyi kullanan eleman ya da elemanların canlı (dinamik) bir listesini döndürür (bu metot, geriye `NodeList` nesnesi döndürür).

Kullanımı: `elemanlistesi = document.getElementsByClassName(classNames)`

İlk durumda; belge içerisinde belirtilen sınıf seçici adını kullanan (yani belirtilen sınıf seçicinin atandığı) elemanların listesini (NodeList) döndürür.

`elemanlistesi = temelEleman.getElementsByClassName(classNames)`

İkinci durumda; belirtilen temel eleman (referans alınan eleman) içerisinde bulunan ve belirtilen sınıf seçici adını kullanan elemanların listesini (`NodeList`) döndürür.

`NodeList` nesnesinin içindeki eleman referanslarına ulaşmak için;

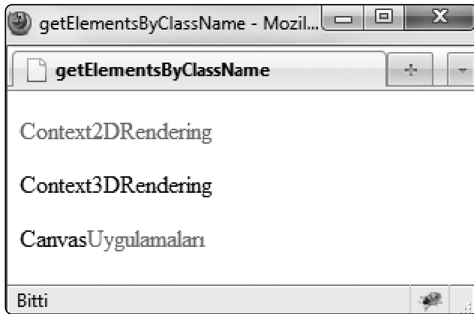
`NodeList[index numarası]` ya da `NodeList.item(index numarası)` yöntemlerinden birini kullanabilirsiniz (`index` numarası `0`'dan başlayacaktır).

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8">
            <title>getElementsByClassName</title>
            <style type="text/css">
            </style>
            <script type="text/javascript">
        var init=function(){
        /*class özelliğine "onem" sınıf seçici değeri atanmış belge içerisindeki tüm
        elemanların Dinamik bir listesine ulaşalım ve bu listeyi bir değişkene atayalım
        (Listenin atandığı değişken artık bir NodeList nesnesi olacaktır)*/
        var elemanlarOnem=document.getElementsByClassName("onem");
        /*NodeList nesnesi (elemanlarOnem) içindeki elemanların referansına
        ulaşalım ve bu elemanlara bir color tanımlaması yapalım*/
          for(var i=0;i<elemanlarOnem.length;i++){
             elemanlarOnem[i].style.color="red";
```

```
            }
          }
                </script>
        </head>
        <body onload=init();>
<div id="ornek">
 <p id="p1" class="onem">Context2DRendering</p>
 <p id="p2" class="hata haber">Context3DRendering</p>
 <p id="p3" class="baslik">Canvas<span class="onem">Uygulamaları</span></p>
</div>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset="utf-8">
                <title>getElementsByClassName</title>
                <style type="text/css">
                </style>
                <script type="text/javascript">
          var init=function(){
          /*class özelliğine "onem" sınıf seçici değeri atanmış belge içerisindeki tüm
          elemanların Dinamik bir listesine ulaşalım ve bu listeyi bir değişkene atayalım
          (Listenin atandığı değişken artık bir NodeList nesnesi olacaktır)*/
          var elemanlarOnem=document.getElementsByClassName("onem");
          /*NodeList nesnesi(elemanlarOnem) içindeki elemanların referansına ulaşalım ve
          bu elemanlara bir color tanımlaması yapalım*/
```

```
            for(var i=0;i<elemanlarOnem.length;i++){
                elemanlarOnem[i].style.color="red";
            }
        }
                </script>
        </head>
        <body onload=init();>
<div id="ornek">
 <p id="p1" class="onem hata">Context2DRendering</p>
 <p id="p2" class="hata onem">Context3DRendering</p>
 <p id="p3" class="baslik">Canvas<span class="onem">Uygulamaları
</span></p>
</div>
        </body>
</html>
```

Elemanların `class` özelliklerine birden fazla sınıf seçici adı arada boşluk olmak kaydıyla yazılabilir. Bu durumda class özelliğine yazılan sınıf seçicilerin sakladıkları CSS tanımlamaları elemanlara uygulanır (Dikkat edin, öncelik ilk yazılan sınıf seçicinindir).

Yukarıdaki örneğimize dönersek, belgedeki bir elemanın class özelliğine atanan sınıf seçici isimleri içerisinde onem sınıf seçici adı varsa bu eleman `getElementsByClassName("onem")` metodunun oluşturduğu `NodeList` listesine eklenir.

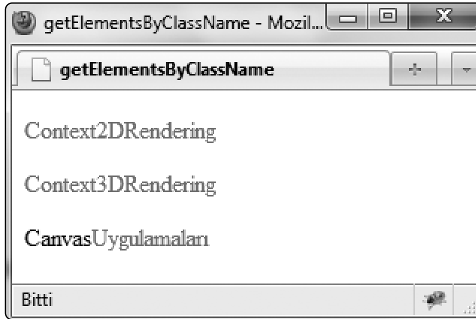
`getElementsByClassName()` metodu için elemana belirtilen sınıf seçicinin atanmış olması yeterlidir. Seçicinin öncelik sırası bu metot için önemli değildir.

`getElementsByClassName("onem")` metodu;

```
 <p id="p1" class="onem hata">Context2DRendering</p>
 <p id="p2" class="hata onem">Context3DRendering</p>
```

Her iki elemanı da geri döndürdüğü `NodeList` listesine ekler.

Ekran görüntüsü:



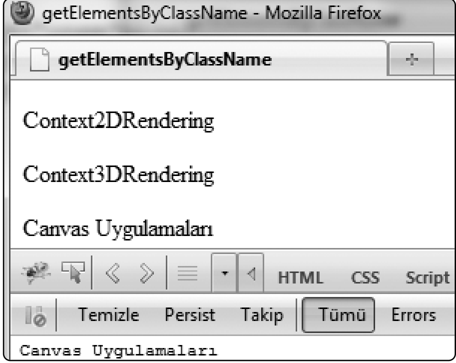
Firefox 3.6 ekran görüntüsü

Örnek:

```
<!DOCTYPE html>
<html>
       <head>
              <meta charset=”utf-8”/>
              <title>getElementsByClassName</title>
              <style type=”text/css”>
              </style>
              <script type=”text/javascript”>
        var init=function(){
        /*class özelliğine “onem baslik”(iki tane sınıf seçici adı) sınıf seçici
          isim listesi atanmış belge içerisindeki tüm elemanların dinamik bir
          listesine ulaşalım ve bu listeyi bir değişkene atayalım(Listenin atandığı
          değişken artık bir NodeList nesnesi olacaktır)*/
        var elemanlarOnem=document.getElementsByClassName(“onem baslik”);
        /*NodeList nesnesi(elemanlarOnem) içindeki elemanların referansına
          ulaşalım ve bu elemanların içeriklerini firebug konsoluna yazdıralım*/
          for(var i=0;i<elemanlarOnem.length;i++){
            console.log(elemanlarOnem[i].textContent);
          }
        }
              </script>
       </head>
       <body onload=init();>
<div id=”ornek”>
 <p id=”p1” class=”onem”>Context2DRendering</p>
 <p id=”p2” class=”baslik”>Context3DRendering</p>
 <p id=”p3” class=”onem baslik”>Canvas Uygulamaları</p>
```

```
</div>
       </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ve Firebug 1.6.2 ekran görüntüsü

Tarayıcı desteği: Internet Explorer 9, Firefox 3+, Opera 9.5+, Safari 3.1+

Seçici Metotlarını Desteklemeyen Tarayıcı Sürümleri için Çözüm

Yukarıda anlatılan metotların hangi tarayıcılar tarafından ne ölçüde desteklendiğini önceki konuda anlatmıştık. Ne yazık ki tarayıcılar W3C bildirimlerini ya da diğer bağlı teknolojileri aynı oranda desteklememektedirler. Bu durum programcıya ek bir yük getirir. Düşünün sayfa tasarlarken bir eleman (nesne) ya da özel bir nesne kullanacaksanız ilk önce tarayıcı desteğine bakıp, daha sonra desteklemeyen tarayıcılar için ek kod yazmanız gerekecektir. Yani farklı tarayıcılar için farklı kodlar yazmanız gerekebilir. Aşağıdaki örneğe bakıp konumuza devam edelim.

Örneğe geçmeden önce...

`firstElementChild` (Bir elemanın içindeki ilk eleman düğümüne ulaşmak için),

`lastElementChild` (Bir elemanın içinde bulunan son eleman düğümüne ulaşmak için),

`nextElementSibling` (HTML ağaç yapısında referans alınan elemandan sonra gelen elemanın (kardeş eleman) referansını almak için),

`previousElementSibling` (HTML ağaç yapısında referans alınan elemandan önce gelen elemanın (kardeş eleman) referansını almak için) kullanılırlar ve bu özellikler element düğümleri (elemanlar) için tanımlıdırlar.

Bu özellikler **Document Object Model Level 2 Traversal and Range** bildirimi ile tanımlanmışlardır. Bu özelliklere tarayıcılar tarafından verilen desteğe bakalım.

Tarayıcı desteği: Internet Explorer 9, Firefox 3.5+, Opera 10.10+, Safari 4+

Şimdi tarayıcıların desteklemeyen sürümleri için bu özellikleri kullanılabilir hale getirelim. Kodlarımızı yazmaya başlayalım.

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="utf-8"/>
      <title>Document Object Model Level 2 Traversal and Range</title>
      <script type="text/javascript">
         var jsharp = {
                /*jsharp javascript Framework 1.0
                 * (Document Object Model Level 2 Traversal and Range
                   partial support)
                 * İbrahim Çelikbilek*/
              _firstElementChild:function(e_ref/*e_ref:elemanReferansı*/){
                  /*Bir elemanın içindeki ilk eleman düğümüne ulaşmak için
                    *_firstElementChild() metodunu tanımladık.Eğer eleman başka
                    bir eleman içermiyorsa _firstElementChild() metodu geriye
                    "null" değerini döndürür.*/
                    if (e_ref.firstElementChild == undefined) {
                        //Eğer firstElementChild özelliği desteklenmiyorsa;
                        //firstChild özelliği kullanılır.
                        //bu özellik elemanın içindeki ilk düğümü(Node)
                        //geriye döndürür.

                        var child = e_ref.firstChild;
                        //e_ref ile tanımlı elemanının ilk çocuk düğümüne ulaştık.

                        while (child) {
                            if (child.nodeType == 1) {
                        /*Bir düğümün türüne bakarak bu düğümün ne düğümü
                        olduğunu  anlayabiliriz.(Element, Text...)
                        * Bir düğümün türüne ulaşmak için "nodeType" özelliği
                        kullanılır.
                        * var type = node.nodeType;
                        * Aşağıda düğüm türleri ve nodeType değerleri
                        verilmiştir;
```

```
        * ELEMENT_NODE = 1(Eğer düğümün nodeType özelliğinin
          değeri 1 ise bu düğüm element düğümüdür.)
        * ATTRIBUTE_NODE = 2
        * TEXT_NODE = 3
        * .......*/

                return child;
              /*e_ref ile tanımlı elemanının çocuk düğümü eğer
                element düğümü ise bu elemenin referansı geriye
                döndürülecek ve döngüden çıkılacak.*/
              }
              else {
                child = child.nextSibling;
              /* e_ref ile tanımlı elemanının ilk çocuk
                düğümü eğer element düğümü değil ise bu
                durumda bu düğümün kardeşi olan diğer
                düğümün referansını nextSibling özelliği ile aldık.
                *Eğer eleman içerisinde kardeş düğüm yoksa
                nextSibling özelliği geriye "null" değerini döndürür.
                Bu değerde while(null){} tanımlamasına
                neden olacağından döngüden çıkılır.*/
              }
          }
      }
      else {
          /* Eğer tarayıcı firstChildElement özelliğini destekliyorsa;
             Kullanımı:
             childNode=elNodeReference.firstElementChild;
             elemanın içindeki ilk elemanının referansını döndürür.*/
             return e_ref.firstElementChild;
      }
      return null;
     /*Eğer eleman içerisinde bir element yoksa ya da
     element düğümü yoksa geriye "null" değerini döndüreceğiz*/
  },

 _lastElementChild: function(e_ref/*e_ref:elemanReferansı*/){
     /*Bir elemanın içinde bulunan son eleman düğümüne
     ulaşmak için_lastElementChild() metodunu tanımladık*/

      if (e_ref.lastElementChild == undefined) {
```

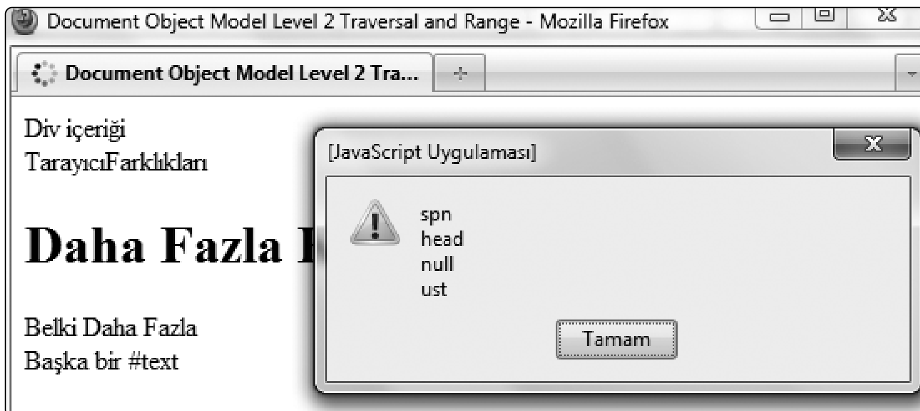
```
                  /* e_ref tanımlı elemanın içindeki en son düğüme ulaştık..*/
        var child = e_ref.childNodes[e_ref.childNodes.length - 1];
                  /* e_ref referansına sahip elemanın içindeki en son
                     elemana ulaşmak için childNodes özelliği kullanıldı.
                     Bu özellik eleman içindeki tüm düğümlerin
                     bir listesini oluşturur.(NodeList)*/
                  while (child) {
                       if (child.nodeType == 1) {
                            return child;
                         /* e_ref ile tanımlı elemanın içindeki en son
                            düğüm eğer element düğümü ise bu
                            elementin referansı geriye döndürüldü.*/
                       }
                       else {
                            child = child.previousSibling;
                          /* e_ref tanımlı elemanın içindeki en son
                             düğüm eğer element düğümü değilse, en
                             son düğümden bir önceki düğüme geçmek
                             (bir önceki düğümün referansını) almak
                             için previousSibling özelliğini kullandık. */
                       }
                  }
             }
             else {
                /* Eğer lastElementChild özelliğini destekliyorsa;
                   Kullanımı:
                   childNode = elNodeReference.lastElementChild;
                   elemanın içindeki son elemanının referansını döndürür.*/
                   return e_ref.lastElementChild;
             }
             return null;
           /*Eğer eleman içererisinde bir element yoksa ya da element
             düğümü yoksa geriye "null" değerini döndüreceğiz */
        },
    _nextElementSibling: function(e_ref/*e_ref:elemanReferansı*/){
           /*
             HTML ağaç yapısında referans alınan elemandan sonra
             gelen elemanın(Kardeş eleman) referansını almak için
             _nextElementSibling()metodunu tanımladık*/

             if (e_ref.nextElementSibling == undefined) {
```

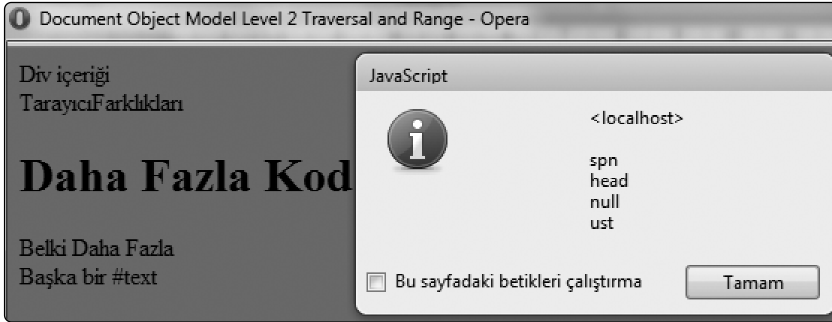
```
                    // nextElementSibling özelliği desteklenmiyorsa...
                    var child = e_ref.nextSibling;
                    while (child) {
                        if (child.nodeType == 1) {
                            return child;
                        }
                        else {
                            child = child.nextSibling;
                        }
                    }
                }
                else {
                    return e_ref.nextElementSibling;
                }
                return null;
            },
    _previousElementSibling: function(e_ref/*e_ref:elemanReferansı*/){
            /* HTML ağaç yapısında referans alınan elemandan önce
               gelen elemanın (kardeş eleman) referansını almak için
               _previousElementSibling metodunu tanımladık*/
                if (e_ref.previousElementSibling == undefined) {
                    //previousElementSibling özelliği desteklenmiyorsa...
                    var child = e_ref.previousSibling;
                    while (child) {
                        if (child.nodeType == 1) {
                            return child;
                        }
                        else {
                            child = child.previousSibling;
                        }
                    }
                }
                else {
                    return e_ref.previousElementSibling;
                }
                return null;
            }
        }
var init = function(){
```

```
        var eleman = document.getElementById("bir");
        var child_element_first = jsharp._firstElementChild(eleman);
        var child_element_last = jsharp._lastElementChild(eleman);
        var nextSiblingElement = jsharp._nextElementSibling(eleman);
        var previousSiblingElement = jsharp._previousElementSibling(eleman);
        alert(child_element_first.id + "\n"+
child_element_last.id+"\n" +nextSiblingElement +"\n"
+previousSiblingElement.id+"\n" );
        /*nextSiblingElement isimli değişkenin değeri null olacaktır. Çünkü div#bir
        elemanından sonra gelen ve kardeşi durumda olan bir eleman düğümü yoktur.*/
             }
        </script>
    </head>
    <body onload="init();">
       <div id="ust">Div içeriği</div>
        <div id="bir">
             Tarayıcı<span id="spn">Farklıkları</span>
             <h1 id="head">Daha Fazla Kod</h1>
             Belki Daha Fazla
        </div>
              Başka bir #text
    </body>
</html>
```

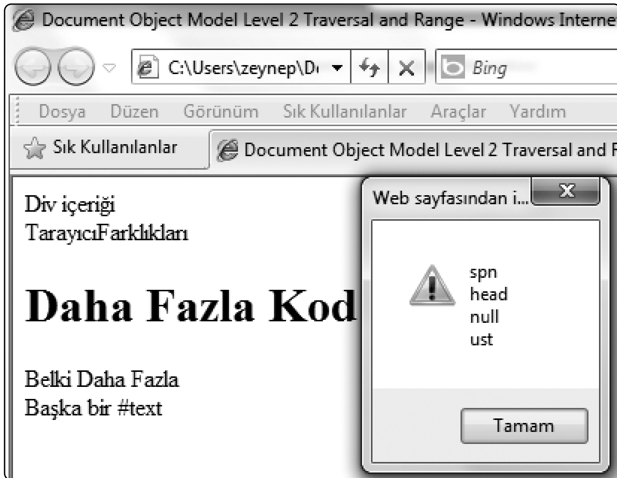
Ekran görüntüleri:



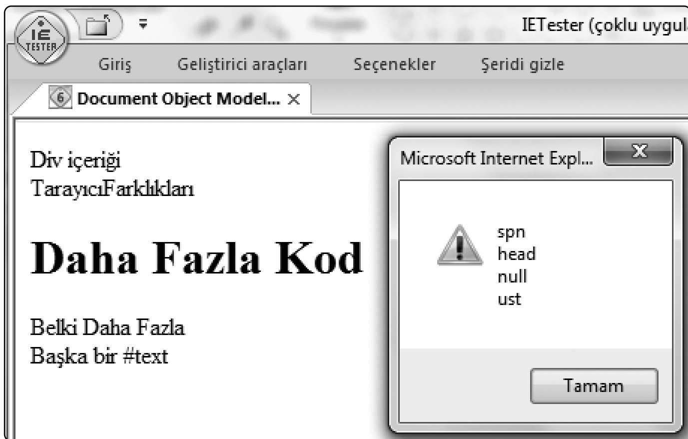
Firefox 3.6 ekran görüntüsü



Opera 11 ekran görüntüsü



IE8 ekran görüntüsü



IE6 ekran görüntüsü
(IETester v0.4.7
ile elde edildi)

Sonuç

DOM Level 2 Traversal and Range bildirimi içerisindeki `firstElementChild`, `lastElementChild`, `nextElementChild`, `previousElementSibling` özelliklerini desteklemeyen tarayıcılar için yazdığımız kodlar yukarıda görülmektedir. Bu durum, yeni başlayanlara için biraz korkutucu gelebilir. Bununla beraber nesne, metot ve özellikleri farklı tarayıcılarda çalıştırmak için kendiniz kod yazmak yerine, var olan hazır JavaScript kütüphanelerini (API) kullanabilirsiniz.

Kullanabileceğiniz JavaScript Kütüphaneleri (API):

- **Dojo:** *http://dojotoolkit.org/*
- **Jquery:** *http://jquery.com/*
- **YUI Library:** *http://developer.yahoo.com/yui/*

`querySelector()`, `querySelectorAll()` ve `getElementsByClassName()` metotlarının tarayıcıların alt sürümleri tarafından desteklenmemesi konusuna geri dönersek; bu metotlar sonuçta CSS seçicileri tarafından hedef alınan eleman ya da elemanların referansını alırlar. Bu metotların yerine Dojo kütüphanesi içinde tanımlı `dojo.query()` metodunu kullanabiliriz. Şimdi aşağıdaki örnekle Dojo kütüphanesinin sayfaya nasıl dahil edileceğini ve belirtilen metodun nasıl kullanılacağını görelim.



Yukarıda belirtilen adresten **Dojo API**'sini indirmek istediğinizde **compressed**, **uncompressed** (açıklama satırları içerir) isimli iki seçenekle karşılaşacaksınız. **Compressed** seçeneğine tıklayarak **dojo.js** (çekirdek yapı) isimli JavaScript dosyasını indirin ve Script etiketini kullanarak sayfanıza dahil edin.

dojo.js içerisinde Dojo kütüphanesinin çekirdek yapısı bulunur. Diğer Dojo modüllerini (*Dijit*, *Dojox*) sayfalarınızda kullanabilmek için Dojo kütüphanesinin tamamını (*Dojo SDK*) indirmeniz gerekir.

Örnek:

```
<!DOCTYPE html>
<html>
      <head>
<meta charset="utf-8"/>
            <title>Dojo</title>
<script src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.js"
type="text/javascript">
```

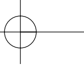
```
        /*Dojo Toolkit 1.5 kütüphanesini bilgisayarınıza indirmeden
         (DOJO kütüphanesini saklayan/yayınlayan
         Sitelerden (CDN)) internet üzerinden kullanabilirsiniz.*/
         </script>
         <script type="text/javascript">
dojo.addOnLoad( function() {
         /*dojo.addOnload(function(){//kodlar}) Dojo kodlarını yazacağımız
         metottur. Bu metot içerisinde Dojo modülleri (Diğer
         dojo>>javaScript dosyaları) ve diğer Dojo nesneleri kullanıma hazırdır.
         DOM nesneleri ve Dojo uygulamaları kullanıma hazır olduktan sonra
         bu metot içerisindeki kodlar çalışır.*/
         /*dojo.query(selectors) bu metod içerisine yazılan CSS
           seçicilerinin hedef aldığı eleman ya da elemanların referansını döndürür.*/
         console.log("id özelliğine 'birP' değeri atanmış eleman**>");
         var elemanP1 = dojo.query("#birP");
         console.log(elemanP1);
         console.log("id özelliğine 'dj' değeri atanmış elemanın
              çocuğu olan tüm elemanlar**>");
         dojo.query("#dj>*").forEach( function(node, index, arr) {
                  /*forEach() metodu sayesinde bir NodeList nesnesi içerisindeki
                    elemanlara tek tek ulaşabiliriz.*/
                  console.log(node);
         });
         var elemanSpn = dojo.query(".spn");
         console.log("class özelliğine 'spn' değeri atanmış eleman**>");
         console.log(elemanSpn);
     var sonP=dojo.query(".onem.bil").forEach(function(node,index,arr){
         console.log("class özelliklerine 'onem bil' şeklinde
         sınıf"+"\n"+" seçici listesi yazılmış elemanlar**>");
         console.log(node);
         });
        console.log("class özelliğine atanan sınıf seçici
listesi içinde 'onem' ya "+"\n"+"da 'bil' değeri bulunan elemanlar**>");
var sonP=dojo.query(".onem,.bil").forEach(function(node,index,arr){
                  console.log(node);
         });
         /*Ayrıca kullanım örneklerine bakın;
         ------------------------------------
         *dojo.query('h3')
         Tüm h3 elemanlarının listesini döndürür(NodeList)
```

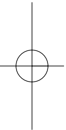
```
                *dojo.query('#main')
                #main id seçicisi ile hedef alınan elemanın referansını döndürür.
                *dojo.query('#main h3')
                id özelliğine 'main' değeri atanmış elemanın soyundan gelen h3
                elemanlarının listesini döndürür.
                *dojo.query('div#main')
                id özelliğine 'main' değeri atanmış div elemanının referansını döndürür.
                *dojo.query('#main div > h3')
                id özelliğine 'main' değeri atanmış elemanının soyundan gelen div
                elemanlarının çocuğu durumunda olan h3 elemanlarının listesini döndürür.
                *dojo.query('.onem')
                class özelliklerine "onem" değeri atanmış elemanların listesini döndürür.
                *dojo.query('.onem.bil')
                class özelliklerine "onem" ve "bil" sınıf seçici isim listesi atanan
                elemanların listesini döndürür.*/
        });
                </script>
        </head>
        <body>
                <div id="dj">
                        <p id="birP">
                                HerYönüyle
                                <span class="onem">HTML5</span>
                        </p>
                        <p>
                                Dojo
                                <span class="spn">Framework</span>
                        </p>
                        <p class="onem bil">
                                Javascript
                        </p>
                </div>
        </body>
</html>
```

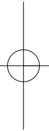
Ekran görüntüsü:

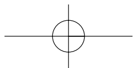
```
Konsol ▾  HTML  CSS  Script  DOM  Net
Temizle  Persist  Takip  Tümü  Errors  Warnings  Info  Debug Info
id özelliğine 'birP' değeri atanmış eleman**>
⊞ [ p#birP ]
id özelliğine 'dj' değeri atanmış elemanın çocuğu olan tüm elemanlar**>
<p id="birP">
<p>
<p class="onem bil">
class özelliğine 'spn' değeri atanmış eleman**>
⊞ [ span.spn ]
class özelliklerine 'onem bil' şeklinde sınıf
seçici listesi yazılmış elemanlar**>
<p class="onem bil">
class özelliğine atanan sınıf seçici listesi içinde 'onem' ya
da 'bil' değeri bulunan elemanlar**>
<span class="onem">
<p class="onem bil">
```

Firefox 3.6 ve Firebug 1.6.2 ekran görüntüsü









HTML5 ve CSS3

4

HTML5 dilinin getirdiği yeni elemanlar ve yapısal özellikler sayesinde CSS kodlarını daha etkili bir şekilde kullanabilir, daha anlamsal CSS seçicileri oluşturabilirsiniz. CSS'i, basamaklı stil sayfaları ya da stil şablonları olarak tanımlayabiliriz. CSS ile belgenizin görünümünü kontrol edebilir. Sayfaya yerleştirdiğiniz içeriklere gelişmiş görsel özellikler tanımlayabilirsiniz.

Sayfanızın sunumunu tek bir yere yazdığınız CSS kodları ile kontrol edebilir ya da oluşturacağınız harici bir CSS dosyası ile birden fazla sayfanın görünümünü kontrol edebilirsiniz.

CSS kuralları, sözdizimi ve özellikleri olan bir yapıdır. Bu yapının CSS1, CSS2 ve en son CSS3 sürümleri çıkarılmıştır. Aşağıda CSS3 seçicilerinin tanımı ve kullanımlarıyla ilgili bilgiler verilmiştir.

CSS3 Selectors

Seçici; aldıkları değerlerle beraber bir grup özelliği temsil eden bir addır. Özellik grupları bu adla temsil edilirler. Bazı durumlarda oluşturduğunuz seçiciler HTML elemanı ya da elemanlarına özelliklerin uygulanmasını sağlamak (belirtilen eleman ya da elemanları hedef almak) için anahtar sözcükler içerebilir.

Şimdi CSS3 yapısı içerisindeki seçicilere bakalım.

Seçici Tipi	Açıklama	Kullanımı	CSS
Type Selectors (Eleman) HTML etiket seçiciler	HTML elemanlarının var olan görsel özelliklerini değiştirmek ya da belirtilen elemanlara yeni görsel özellikler eklemek için kullanılırlar. Seçici ismi olarak HTML eleman isimleri kullanılır.	**h1**{bildirimler} Sayfadaki tüm h1 elemanlarını hedef alır.	CSS1
ID Selectors (#) ID seçiciler	Kimlik seçicileri, HTML elemanlarının id özelliklerine atanan benzersiz isimler seçici adı olarak kullanılır.	**#onemli**{bildirimler} id özelliğine "onemli" değeri atanmış elemanı hedef alır.	CSS1
Class Selectors (.) Sınıf seçiciler	Bir HTML elemanı için farklı CSS özellik grupları ya da sayfadaki tüm HTML elemanları tarafından kullanılabilecek CSS özellik grupları tanımlayabilirsiniz.	**.baslik**{bildirimler} class özelliğine "baslik" değeri atanmış elemanları hedef alır. -------------------------------- **p.baslik**{bildirimler} p elemanları içerisinden class özelliğine "baslik" değeri atanmış elemanları hedef alır.	CSS1
Universal Selector (*) Evrensel seçiciler	Body ya da temel bir eleman içerisindeki tüm elemanlara CSS tanımlamaları yapmak için kullanılırlar. Evrensel seçici "*" (yıldız) karakteri ile oluşturulur.	**#anaListe** *{bildirimler} id özelliğine "anaListe" değeri atanmış elemanın içindeki tüm elemanlar hedef alınır.	CSS2
Descendant Selectors () Soy (torun seçiciler) seçiciler	HTML elamanlarının oluşturduğu soy ağacı yapısından faydalanarak belirlediğiniz bir elemanın soyundan gelen eleman ya da eleman gruplarına CSS kodları atamak için kullanılır.	**div#content** p{bildirimler} id özelliğine "content" değeri atanmış elemanın soyundan gelen (içinde olan) tüm p elemanları hedef alınır.	CSS1
Child Selector (>) Çocuk seçiciler	Bir HTML elemanın çocuğu durumdaki eleman ya da elemanlara CSS özellikleri atamak için kullanılır. Torun seçicilere benzemekle beraber daha özel durumu olarak düşünebilirsiniz.	**div#content>**p{bildirimler} id özelliğine "content" çocuğu durumunda olan tüm p elemanları hedef alınır.	CSS2

Adjacent Sibling Selector (E1+E2) Bitişik kardeş seçiciler	Aynı ebeveyn içerisinde bulunan (kardeş olan) belirlediğimiz elemandan hemen sonra gelen elemana CSS kodları tanımlamak için kullanılır.	**li#acil+li**{bildirimler} id özelliğine “acil” değeri atanmış li elemanından sonra gelen ve kardeşi durumunda olan li elemanı hedef alınır. -------------------------------- <ul id=”bir”> <li id=”acil”>1. eleman 2.eleman 3.eleman 4.eleman 5.elaman 	CSS2
General Sibling Selectors (E1 ~ E2) Genel kardeş seçiciler	Aynı ebeveyn içerisinde bulunan (kardeş olan) belirlediğimiz elemandan sonra gelen elemana/elemanlara CSS kodları atamak için kullanılırlar.	**li#acil~li**{bildirimler} id özelliğine “acil” değeri atanmış li elemanından sonra gelen (kardeşi durumunda olan) li elemanları hedef alınır. -------------------------------- <ul id=”bir”> <li id=”acil”>1. eleman 2.eleman 3.eleman 4.eleman 5.elaman 	CSS3

Attrıbute Selectors (Özellik Seçiciler)

HTML etiketlerinde kullanılan özelliklere, bu özelliklerin aldıkları değerlere ya da özelliklere atanan değerlerin kısmi bir bölümünün konumuna göre seçiciler oluşturabilirsiniz.

Seçici Tipi	Açıklama	Kullanımı	CSS
[attribute]	Belirtilen özelliği kullanmış eleman ya da elemanlara CSS kodları tanımlamak için kullanılır.	**div#init p[title]** **{bildirimler}** id özelliğine "init" değeri atanmış div elemanının soyundan gelen title özelliği tanımlanmış tüm p elemanlarını hedef alır. -------------------------------- <div id="init"> <p title="aciklama"> XHTML ve CSS Kitabı </p> <p>CSS Başvuru Kaynağı </p> <p title="son">Geniş Açıklama ve Örneklerle </p> </div>	CSS2
[attribute="value"]	Köşeli parantezler içerisinde bir özellik ve değer tanımlaması yapılarak seçici oluşturulur. Bu durumda belirtilen özelliği değeri ile beraber kullanan etiketlere CSS kodları uygulanır.	**div#init p[title="son"]** **{bildirimler}** id özelliğine "init" değeri atanmış div elemanının soyundan gelen ve title özelliğine "son" değeri atanmış p elemanını hedef alır. -------------------------------- <div id="init"> <p title="aciklama"> XHTML ve CSS Kitabı </p> <p>CSS Başvuru Kaynağı </p> <p title="son">Geniş Açıklama ve Örneklerle </p> </div>	CSS2

[attribute~="value"]	HTML elemanı belirtilen özelliği (attribute) kullanmış ve bu özelliğe aldığı değer listesi içerisinde bizim tanımladığımız metin (value) var ise bu elemana belirtilen CSS kodları uygulanır. *Değer listesi boşluk içeren birden fazla metnin oluşturduğu string dize.	**p[title~="html5"]** {bildirimler} title özelliğine aldığı değer listesi içerisinde "html5" metni bulunan p elemanını hedef alır. -------------------------------- <p title="csshtml5"> XHTML ve CSS Kitabı</p> <p title="css html5"> CSS Başvuru Kaynağı</p>	CSS2
[attribute*="value"]	HTML elemanı belirtilen özelliği (attribute) kullanmış ve bu özelliğe aldığı değer içerisinde bizim tanımladığımız metin (value) var ise bu elemana belirtilen CSS kodları uygulanır.	**p[title~="html5"]** {bildirimler} title özelliğine aldığı değer içerisinde "html5" metni bulunan p elemanlarını hedef alır. -------------------------------- <p title="csshtml5"> XHTML ve CSS Kitabı</p> <p title="css html5"> CSS Başvuru Kaynağı</p>	CSS3
[attribute^="value"]	Bir HTML elemanının belirtilen özelliğine (attribute) aldığı değerin başında tanımladığımız değer (value) var ise bu HTML elemanına CSS kodları uygulanır.	**p[title^="html5"]** {bildirimler} title özelliğine aldığı değerin başında "html5" metni bulunan p elemanlarını hedef alır. -------------------------------- <p title="html5css"> XHTML ve CSS Kitabı </p> <p title="css html5">CSS Başvuru Kaynağı</p>	CSS3
[attribute$="value"]	Bir HTML elemanının belirtilen özelliğine (attribute) aldığı değerin sonunda tanımladığımız değer (value) var ise bu HTML elemanına CSS kodları uygulanır.	**p[title$="html5"]** {bildirimler} title özelliğine aldığı değerin sonunda "html5" metni bulunan p elemanlarını hedef alır. --------------------------------	CSS3

<table>
<tr><td></td><td></td><td><p title="html5css"> XHTML ve CSS Kitabı </p>
<p title="css html5">CSS Başvuru Kaynağı</p></td><td>CSS3</td></tr>
<tr><td>[attribute|="value"]</td><td>[attribute^="value"] seçicisi kalıbı ile benzerdir (<code>value</code> değeri özelliğe alınan değerin başında olmalı). Fakat <code>value</code> değerinden sonra – (tire) karakteri kullanılmış olmalıdır.</td><td>p[title|="html5"] {bildirimler}
title özelliğine aldığı değerin başında "html5-" metni bulunan p elemanlarını hedef alır.

<p title="html5-css"> XHTML ve CSS Kitabı</p>
<p title="html5css">CSS Başvuru Kaynağı</p>
<p title="html5 css"> javascript</p></td><td>CSS2</td></tr>
<tr><td>[ns|attribute]</td><td>Namespace (isim alanı) için özellik seçiciler oluşturulabilir.</td><td></td><td>CSS3</td></tr>
</table>

Pseudo-elements (Sözde Elemanlar)

Aslında bir HTML elemanı değillerdir, fakat bir HTML elemanı gibi davranırlar.

Seçici Tipi	Açıklama	CSS
::after, :after	Bir elemandan sonra içerik oluşturmak için kullanılır. (Satır içi bir content oluşturulur.) • İçeriği tanımlamak için `content` özelliği kullanılır. **CSS 2 Söz Dizimi:** `element:after {bildirimler}` **CSS 3 Söz Dizimi:** `element::after {bildirimler}` • İki tanımlamada aynı işlemi yapar. CSS3 bu özelliği sözde sınıf ve sözde elemanlar arasında söz dizimi farkı oluşturmak için **`::after`** şeklinde değiştirmiştir.	CSS2
::before, :before	Bir elemandan önce içerik oluşturmak için kullanılır. (Satır içi bir content oluşturulur.) • İçeriği tanımlamak için `content` özelliği kullanılır.	CSS2

	CSS 2 Söz Dizimi: `element:before{bildirimler}` **CSS 3 Söz Dizimi:** `element::before {bildirimler}` • iki tanımlamada aynı işlemi yapar. CSS3 bu özelliği sözde sınıf ve sözde elemanlar arasında söz dizimi farkı oluşturmak için **`::before`** şeklinde değiştirmiştir.	
`:first-letter, ::first-letter`	Metnin ilk harfine stil tanımlaması yapmak için kullanılır. CSS3 bu özelliği sözde sınıf ve sözde elemanlar arasında söz dizimi farkı oluşturmak için **`::first-letter`** şeklinde değiştirmiştir.	CSS1
`:first-line, ::first-line`	Metnin ilk satırına stil tanımlaması yapmak için kullanılır. CSS3 bu özelliği sözde sınıf ve sözde elemanlar arasında söz dizimi farkı oluşturmak için **`::first-line`** şeklinde değiştirmiştir.	CSS1
`::selection`	Bir elemana kullanıcı tarafından seçilme durumunda uygulanacak CSS tanımlamaları yapmak için kullanılır. Bu sözde sınıf içerisinde sadece `color`, `background`, `cursor`, `outline` özellikleri tanımlanabilir.	CSS3

Link and user action pseudo-classes (Link ve Kullanıcı Eylemleri için Sözde Sınıflar)

Seçici Tipi	Açıklama	CSS
`:link, :visited`	`link`, `visited` sözde sınıfları sadece <a> elemanı için tanımlanabilir. **`:link`** Ziyaret edilmemiş linklerin style tanımlamasını yapmak için kullanılan sözde sınıftır. **`:visited`** Kullanıcı tarafından ziyaret edilmiş yani tıklanmış linklerin style tanımlamasını yapmak için kullanılan sözde sınıftır.	CSS 1/2
`:active`	Bir elemanın mouse ile üzerine tıklandığı andaki stil tanımlamasını yapmak için kullanılır.	CSS 1/2
`:focus`	Bir input elemanın üzerine odaklanıldığında (Eleman klavye ya da Mouse ile seçildiğinde) kullanılacak CSS özellik kodlarını tanımlar.	CSS2
`:hover`	Bir elemana Mouse ile üzerine gelindiği anda uygulanacak stil tanımlamalarını yapmak için kullanılır.	CSS2

Language pseudo-class :lang

Elemanların lang özelliklerine aldıkları dil koduna göre seçiciler oluşturulur.

Kullanımı:

```
div:lang(tr){
Bildirimler
}
```

lang özelliğine "tr" değeri alan div elemanlarını hedef alır.

```
<div lang="tr">Birinci Div</div>
<div lang="en-US">İkinci Div</div>
<div lang="en">Üçüncü Div</div>
```

CSS Bildirimi: CSS 2

:target

Sayfa içi linke tıklandığında tarayıcı tarafından hedef alınan çapa elemanı (Hedef Eleman) için geçerli olacak CSS tanımlamaları yapar.

Kullanımı:

```
h1:target{Bildirimler}
```

```
<a href="#hedef1">Birinci Başlık</a>
<!--Diğer Elemanlar -->
<h1 id="hedef1">Buradasınız
</h1>
```

Kullanıcı linke tıklandığında CSS tanımlamaları h1 elemanına uygulanır.

CSS Bildirimi: CSS 3

Negation pseudo-class :not

Tanımlanan Seçici tarafından hedef alınmayan elemanlara CSS tanımlaması yapmak için kullanılır.

Kullanımı:

```
p:not(.onem){
Bildirimler}
```

p elemanları içerisinden ".onem" sınıf seçicisini kullanmayanları hedef alır.

```
<p class="onem"> XHTML ve CSS Kitabı</p>
<p class="onem">CSS Başvuru Kaynağı</p>
<p class="duyur"> javascript</p>
```

CSS Bildirimi: CSS 3

Structural pseudo-classes (Yapısal Sözde Sınıflar)

```
:root                  :nth-last-of-type( )   :last-of-type
:nth-child( )          :first-child           :only-of-type
:nth-of-type( )        :last-child            :only-child
:nth-last-child( )     :first-of-type         :empty
```

Aşağıda ayrıntılı bir şekilde anlatılmıştır.

UI States pseudo-classes (Kullanıcı Arayüzü için Sözde Sınıflar)

Seçici Tipi	Açıklama	Kullanımı	CSS
`:checked`	`radio`, `checkbox` ve `menu` öğeleri kullanıcı tarafından seçildiğinde belirtilen elemanlar için aktif olacak CSS tanımlamaları yapabilirsiniz.	**`input[type="checkbox"]:checked`** `{Bildirimler}` type özelliğine "checkbox" değeri atanan 2. input elemanı kullanıcı tarafından seçildiğinde (işaretlendiğinde) belirtilen CSS kodları elemana uygulanır. -------------------------------- `<input type="radio"/>` `<input type="checkbox"/>`	CSS3
`:enabled, :disabled`	**`:enabled`** aktif, kullanılabilir (`disabled="disabled"` tanımlaması yapılmamış) form elemanlarına CSS kodları tanımlamak için kullanılır. **`:disabled`** Pasif (`disabled="disabled"` tanımlaması yapılmış) form elemanlarına CSS kodları tanımlamak için kullanılır.	**`input[type="text"]:enabled`** `{Bildirimler}` `type` özelliğine "text" değeri atanmış kullanılabilir input elemanlarını hedef alır. -------------------------------- **`input[type="text"]:disabled`** `{Bildirimler}` type özelliğine "text" değeri atanmış pasif input elemanlarını hedef alır. -------------------------------- `<!—Birinci Seçicinin Hedefi-->` `<input type="text"/>` `<!—İkinci Seçicinin Hedefi-->` `<input type="text" disabled="disabled"/>`	CSS3

:default	Kullanıcıya birden fazla seçenek arasından seçim yapma imkanı veren HTML elemanlarının (menu elemanları, açılan kutular, Listeler) içinde liste yapısını oluşturan ve varsayılan olarak seçili bulunan alt elemanlara CSS tanımlamaları yapmak için kullanılır.	**option:default** {Bildirimler} açılan kutu yapısı (Select Elemanı) içerisinde varsayılan olarak seçili ilk option elemanı hedef alınır. -------------------------------- <select> <option selected=”selected”> Seçenek_1/option> <option>seçenek_2</option> <option>seçenek_3</option> </select>	CSS3
:invalid	Bir form elemanına girilen değer elemana girilebilecek veri formatında değilse, yani doğrulama sınamasında hata olursa belirtilen elemana CSS kodları uygulanır.	**input[type=”email”]:invalid** {Bildirimler} type özelliğine “email” değeri atanmış input elemanına doğru bir email adresi yazılmamış ise CSS kodları elemana uygulanır. -------------------------------- <input type=”email”/> <input type=”text”/>	CSS3
:optional	Form eleman gurubu içerisinde isteğe bağlı veri girişi yapılan elemanlara CSS tanımlaması yapmak için kullanılır.	**input:optional{Bildirimler}** input elemanları içerisinden required (içerik yazmak ya da seçim yapmak gerekli) tanımlaması yapılmayan 1. ve 2. form elemanları hedef alınır. -------------------------------- <input type=”text”/> <input type=”checkbox”/> <input type=”email” required=”required” />	CSS3
:required	Required (içerik yazmak ya da seçim yapmak gerekli) özelliğini kullanılmış form elemanlarına CSS tanımlamaları yapmak için kullanılır.	**input:required{Bildirimler}** input elemanları içerisinden required tanımlaması yapılmış sonuncu form elemanı hedef alınır. -------------------------------- <input type=”text”/> <input type=”checkbox”/> <input type=”email” required=”required” />	CSS3

Page pseudo-classes (Sayfa için sözde sınıflar)

Seçici Tipi	Açıklama	CSS
:first, :left, :right	Bu sözde sınıflar belge yazdırılırken geçerli olacak stil tanımlamaları yapmak için kullanılırlar. Bu sözde sınıfların içerisinde sadece belge için **margin**, **orphans**, **windows** ve **page break** tanımlamaları yapılabilir.	CSS3

Structural Pseudo-Classes'ın (Yapısal Sözde Sınıflar) İncelenmesi

Şimdi Structural Pseudo-Classes'ın yapısal özelliklerini inceleyelim.

:nth-child( )

Kapsayıcı eleman içerisindeki pozisyonlarına (kardeş elemanlar listesindeki index numaralarına) göre çocuk eleman ya da elemanlara CSS tanımlamaları yapabilirsiniz.

Kullanımı: `eleman:nth-child(index) { Bildirimler }`

`index` değeri seçici tarafından hedef alınacak çocuk eleman ya da elemanların sıralamadaki `index` numarasını tanımlamak için kullanılır. Bir sayı (*1, 2...*) formül (*n, 2n+1...*) ya da ön tanımlı bir kelime (`odd`, `even`) olabilir.

Sıralamadaki ilk elemanın `index` değeri `1`'den başlar. (*2n+1*) formülü yerine `odd` ve (*2n*) formülü yerine even anahtar kelimelerini kullanabilirsiniz. Formül oluşturulurken sayıcı olarak *n* değişkeni kullanılır (*n=0,1,2...*) ve formül *an+b* formatında yazılır. *a*, *b* değerleri birer tamsayıdır.

Örnek formüller: (*0n+1*), (*-n+3*), (*3n+1*)

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8"/>
            <title>nth-child()</title>
            <style type="text/css">
           div#cont>p:nth-child(1){
           /*Bu seçicinin, div#content elemanının çocuğu olan  ve kardeş elemanlar
```

```
          listesinde 1. sırada olan  p elemanını hedef alması beklenir. Fakat div#content
          elemanı içerisinde  kardeş elemanlar listesinde 1 index numarasına sahip
          p elemanı olmadığından bu seçici herhangi bir işlem yapmaz.*/
          background-color:blue;
        }
        div#cont>p:nth-child(2){
        /*div#content elemanının çocuğu olan ve kardeş eleman sıralamasında
          2. sırada olan p elemanını hedef alır. Yani div elemanının 2 inci çocuğu
          olan  p elemanına css bildirimi uygulanır.*/
        background-color:red;
        }
          </style>
    </head>
    <body>
          <div id="cont">
                <h3>Başlık 1</h3>
                <p>Başlık 1 içerik</p>
                <h3>Başlık 2</h3>
                <p>Başlık 2 içerik</p>
          </div>
    </body>
</html>
```

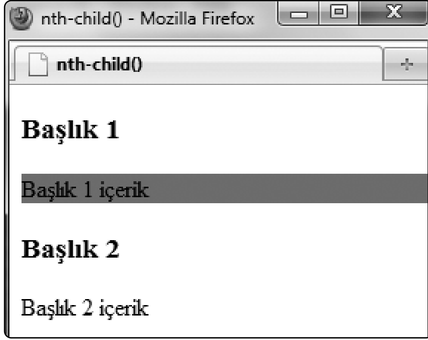
Kardeş elemanlar listesi:

```
<body>
  <div id="cont">
      <h3>Başlık 1</h3>
      <p>Başlık 1 içerik</p>        Kardeş Elemanlar Listesi
      <h3>Başlık 2</h3>
      <p>Başlık 2 içerik</p>
  </div>
</body>
```

`div#cont` elemanı içerisinde bulunan (kardeş elemanlar listesindeki) çocuk elemanlara `nth-child()` seçicisi tarafından tanımlanan index numaralarına bakalım. Dikkat ederseniz index numaraları sayfa akışındaki yerlerine göre tip ayrımı yapılmadan verilmiş.

```
<div id="cont">
  1 <h3>
  2 <p>
  3 <h3>
  4 <p>
</div>
```

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>nth-child()</title>
              <style type="text/css">
                     div#cont>p:nth-child(n+1){
                     background-color:red;
                     }
              </style>
       </head>
       <body>
              <div id="cont">
                     <h3>Başlık 1</h3>
                     <p>Başlık 1 içerik</p>
                     <h3>Başlık 2</h3>
                     <p>Başlık 2 içerik</p>
              </div>
       </body>
</html>
```

Örnekte `index` değeri olarak bir formül yazılmış bu durumda `n` değişkeninin değeri `0`'dan başlayacaktır.

Formülümüz: `p:nth-child(n+1)`

`n=0 >>p:nth-child(1):` Bu durumda `1` index numarasına sahip `p` elemanı hedef alınır. (Böyle bir eleman yok.)

`n=1 >>p:nth-child(2):` Bu durumda `2` index numarasına sahip `p` elemanı hedef alınır.

`n=2 >>p:nth-child(3):` Bu durumda `3` index numarasına sahip `p` elemanı hedef alınır. (Böyle bir eleman yok)

`n=3 >>p:nth-child(4):` Bu durumda `4` index numarasına sahip `p` elemanı hedef alınır.

Yani bu durumda **Kardeş Elemanlar Listesi**'nde 2. ve 4. sırada bulunan `p` elemanlarına CSS kodu uygulanır.

Örnek:

```
<!DOCTYPE html>
<html>
	<head><meta charset="utf-8"/>
		<title>nth-child()</title>
		<style type="text/css">
			ul li:nth-child(2n+1) {
				background-color:blue;
			}
		</style>
	</head>
	<body>
		<ul>
			<li>Bir</li>
			<li>İki</li>
			<li>Üç</li>
			<li>Dört</li>
			<li>Beş</li>
		</ul>
	</body>
</html>
```

Seçiciyi incelersek `ul` elemanı içerisinde bulunan `1`, `3` ve `5` numaralı `li` elemanlarının hedef alındığını göreceğiz.

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

:NTH-OF-TYPE()

Kapsayıcı eleman içerisindeki pozisyonlarına (aynı tipteki kardeş elemanlar listesindeki index numaralarına) göre çocuk eleman ya da elemanlara CSS tanımlamaları yapabilirsiniz. `:nth-child()` seçicisinden farklı olarak aynı tipteki elemanlar ayrı ayrı kardeş elemanlar listesi oluşturur.

```
eleman:nth-of-type(index) { Bildirimler }
```

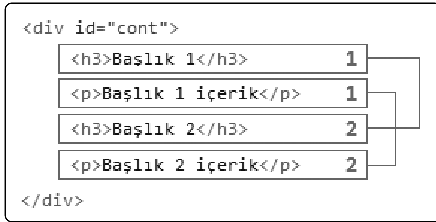
`index` değeri seçici tarafından hedef alınacak çocuk eleman ya da elemanların sıralamadaki index numarasını tanımlamak için kullanılır. Bir sayı (*1, 2...*), formül (*n, 2n+1...*) ya da ön tanımlı bir kelime (`odd`, `even`) olabilir. Çocuk elemanlar içinde bulunan bir türe ait ilk elemanın `index` numarası `1`'dir. (*2n+1*) formülü yerine `odd` ve (*2n*) formülü yerine `even` anahtar kelimelerini kullanabilirsiniz. Formül oluşturulurken sayıcı olarak *n* değişkeni kullanılır (*n=0, 1, 2...*) ve formül `an+b` formatında yazılır. `a`, `b` değerleri birer tamsayıdır. Örnek formüller: (*0n+1*), (*3n+1*)

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>nth-of-type()</title>
              <style type="text/css">
             div#cont>p:nth-of-type(1){
             /*Bu seçici div#content elemanının çocuğu olan ve aynı tipteki kardeş
               elemanlar listesinde 1. sırada olan p elemanını hedef alır.*/
              background-color:blue;
             }
             div#cont>h3:nth-of-type(1){
```

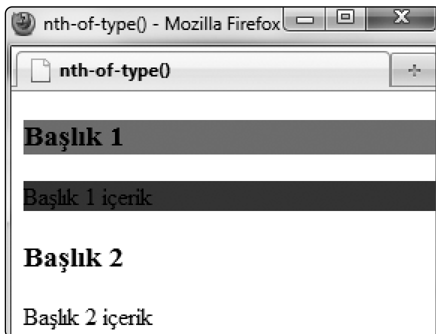
```
            /*div#content elemanının çocuğu olan ve aynı tipteki kardeş eleman
              sıralamasında 1. sırada olan h3 elemanını hedef alır.*/
              background-color:red;
            }
              </style>
       </head>
       <body>
              <div id="cont">
                     <h3>Başlık 1</h3>
                     <p>Başlık 1 içerik</p>
                     <h3>Başlık 2</h3>
                     <p>Başlık 2 içerik</p>
              </div>
       </body>
</html>
```

Aynı tipteki (türdeki) kardeş elemanlar listesi:



Dikkat ederseniz `nth-of-type()` seçicisi her tür (eleman) için ayrı bir kardeş elemanlar listesi tanımlar. `index` numaraları elemanların sayfa akışındaki yerlerine göre tip ayrımı yapılarak verilir.

Ekran görüntüsüne bakalım.

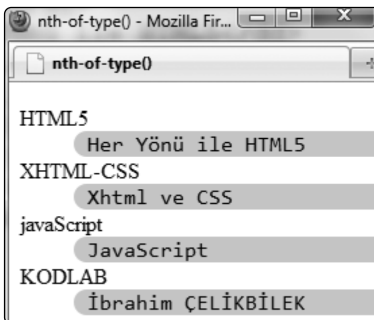


Firefox 3.6 ekran görüntüsü

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>nth-of-type()</title>
              <style type="text/css">
              dd:nth-of-type(n+1){
                font-family:consolas;
                background-color: lightblue;
                text-indent:10px;
                border-radius: 20px;/*CSS3*/
                -moz-border-radius:20px;/*firefox 3.6*/
                -webkit-border-radius:20px;/*Safari 3.0*/
              }
              </style>
       </head>
       <body>
<dl>
  <dt>HTML5</dt>
  <dd>Her Yönü ile HTML5</dd>
  <dt>XHTML-CSS</dt>
  <dd>Xhtml ve CSS</dd>
  <dt>javaScript</dt>
  <dd>JavaScript</dd>
  <dt>KODLAB</dt>
  <dd>İbrahim ÇELİKBİLEK</dd>
</dl>
       </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

`dd:nth-of-type(n+1)` seçicisini incelersek...

n=0 >> dd:nth-of-type(1) Bu durumda 1 index numarasına sahip dd elemanı hedef alınır.

n=1 >> dd:nth-of-type(2) Bu durumda 2 index numarasına sahip dd elemanı hedef alınır.

n=2 >> dd:nth-of-type(3) Bu durumda 3 index numarasına sahip dd elemanı hedef alınır.

n=3 >> dd:nth-of-type(4) Bu durumda 4 index numarasına sahip dd elemanı hedef alınır.

n sayıcısı 4 olduğunda 5 index numarasına sahip dd elemanı aranacaktır. dl elemanı içerisinde en büyük index numarasına sahip dd elemanı 4 olduğundan sayma işlemi durdurulur.

dl elemanı içerisindeki elemanlara `nth-of-type()` seçicisi tarafından tanımlanan index numaralarına görsel olarak bakalım.

```
<dl>
        <dt>HTML5</dt>(dt:1)
        <dd>Her Yönü ile HTML5</dd>(dd:1)
        <dt>XHTML-CSS</dt>(dt:2)
        <dd>Xhtml ve CSS</dd>(dd:2)
        <dt>javaScript</dt>(dt:3)
        <dd>JavaScript</dd>(dd:3)
        <dt>KODLAB</dt>(dt:4)
        <dd>İbrahim ÇELİKBİLEK</dd>(dd:4)
</dl>
```

:NTH-LAST-CHILD( )

Kapsayıcı eleman içerisindeki pozisyonlarına (kardeş elemanlar listesindeki index numaralarına) göre çocuk eleman ya da elemanlara CSS tanımlamaları yapabilirsiniz. `:nth-child()` seçicisinden farklı olarak son çocuğun index numarası 1'dir. Index numaraları en son elemandan başlanarak yukarı doğru arttırılarak verilir.

Kullanımı: `eleman:nth-last-child(index) { Bildirimler }`

index değeri seçici tarafından hedef alınacak çocuk eleman ya da elemanların sıralamadaki index numarasını tanımlamak için kullanılır. Bir sayı (*1, 2...*) formül (*n*,

2n+1...) ya da ön tanımlı bir kelime (odd, even) olabilir. (*2n+1*) formülü yerine odd ve (*2n*) formülü yerine even anahtar kelimelerini kullanabilirsiniz. Formül oluştururken sayıcı olarak n değişkeni kullanılır (*n=0, 1, 2...*) ve formül an+b formatında yazılır. a, b değerleri birer tamsayıdır. Örnek formüller: (*0n+1*), (*-n+4*), (*3n+1*)

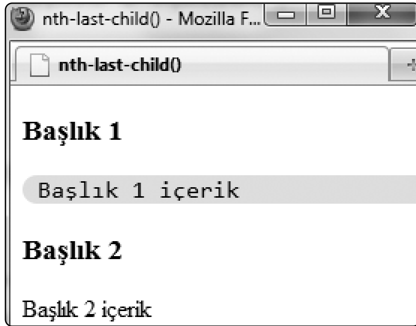
Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>nth-last-child()</title>
              <style type="text/css">
              p:nth-last-child(3){
             /*Kardeş eleman listesinde 3.index numarasına sahip p elemanını
              hedef alır.*/
                font-family:consolas;
                background-color:khaki;
                text-indent:10px;
                border-radius: 20px;/*CSS3*/
                -moz-border-radius:20px;/*firefox 3.6*/
                -webkit-border-radius:20px;/*Safari 3.0*/
              }
              </style>
       </head>
       <body>
        <div id="cont">
              <h3>Başlık 1</h3>
              <p>Başlık 1 içerik</p>
              <h3>Başlık 2</h3>
              <p>Başlık 2 içerik</p>
        </div>
       </body>
</html>
```

Kardeş elemanlar listesi ve `nth-last-child()` seçicisi tarafından elemanlar için tanımlanan index numaralarına bakalım.

```
<div id="cont">
    <h3>Başlık 1</h3>          4
    <p>Başlık 1 içerik</p>     3
    <h3>Başlık 2</h3>          2
    <p>Başlık 2 içerik</p>     1
</div>
```

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

:NTH-LAST-OF-TYPE( )

Kapsayıcı eleman içerisindeki pozisyonlarına (aynı tipteki kardeş elemanlar listesindeki index numaralarına) göre çocuk eleman ya da elemanlara CSS tanımlamaları yapabilirsiniz. `:nth-of-type` seçicisinden farklı olarak bir türün son çocuk elemanının `index` numarası `1`'dir. `index` numaraları aynı tipteki en son elemandan başlanarak yukarı doğru arttırılarak verilir.

```
eleman:nth-of-type(index) { Bildirimler }
```

`index` değeri seçici tarafından hedef alınacak çocuk eleman ya da elemanların sıralamadaki index numarasını tanımlamak için kullanılır. Bir sayı (*1, 2...*) formül (*n, 2n+1...*) ya da ön tanımlı bir kelime (`odd`, `even`) olabilir. (*2n+1*) formülü yerine `odd` ve (*2n*) formülü yerine `even` anahtar kelimelerini kullanabilirsiniz. Formül oluşturulurken sayıcı olarak `n` değişkeni kullanılır (*n=0, 1, 2...*) ve formül `an+b` formatında yazılır. `a`, `b` değerleri birer tamsayıdır.

Örnek formüller: (*0n+1*), (*3n+1*)

Örnek:

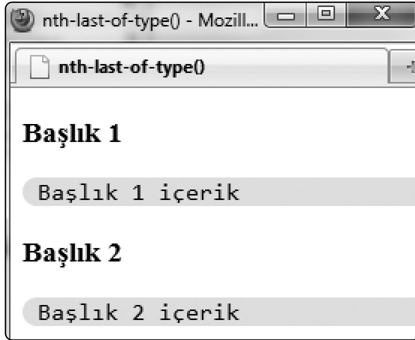
```
<!DOCTYPE html>
<html>
	<head><meta charset="utf-8"/>
		<title>nth-last-of-type()</title>
		<style type="text/css">
		Div#cont>p:nth-last-of-type(n+1){
		  font-family:consolas;
		  background-color:khaki;
		  text-indent:10px;
		  border-radius: 20px;/*CSS3*/
		  -moz-border-radius:20px;/*Firefox 3.6*/
		  -webkit-border-radius:20px;/*Safari 3.0*/
		}
		</style>
	</head>
	<body>
	 <div id="cont">
			<h3>Başlık 1</h3>
			<p>Başlık 1 içerik</p>
			<h3>Başlık 2</h3>
			<p>Başlık 2 içerik</p>
		</div>
	</body>
</html>
```

Aynı türdeki kardeş elemanlar listesi ve `nth-last-of-type()` seçicisi tarafından elemanlar için tanımlanan index numaralarına bakalım.

```
<div id="cont">
    <h3>Başlık 1</h3>        2
    <p>Başlık 1 içerik</p>   2
    <h3>Başlık 2</h3>        1
    <p>Başlık 2 içerik</p>   1
</div>
```

Bu durumda `1`, `2` `index` numaralarına sahip p elemanlarına CSS kodları atanır.

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

:FIRST-CHILD VE :LAST-CHILD

Kapsayıcı eleman içerisindeki ilk çocuk elemana CSS tanımlaması yapmak için `:first-child`, son çocuk elemana CSS tanımlaması yapmak için `:last-child` sözde sınıfları kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset=”utf-8”/>
            <title>first-child,last-child</title>
            <style type=”text/css”>
            div#cont h3:first-child{
            /*div#cont elemanı içerisindeki ilk çocuk durumunda olan
            h3 elemanı hedef alınmıştır.*/
              font-family:consolas;
              background-color:blue;
              text-indent:10px;
             -moz-box-shadow: 0 0 1em blue;/*Firefox 3.5*/
             -webkit-box-shadow: 0 0 1em blue;/*Safari 3.0*/
             box-shadow: 0 0 1em blue;/*CSS3*/
            }
            div#cont p:first-child{
            /*Bu seçici ile div#cont elemanı içerisindeki ilk çocuk durumunda olan p
            elemanı hedef alınmak istenmiştir. Fakat belirtilen eleman içerisinde
            ilk çocuk durumunda olan eleman p değil, h3 elemanıdır. Bu seçici
            herhangi bir işlem yapmaz*/
```

```
        background-color:red;
      }
      div#cont p:last-child{
      /*div#cont elemanı içerisindeki son çocuk durumunda olan
      p elemanı hedef alınmıştır.*/
        font-family:consolas;
        background-color:black;
        text-indent:10px;
        color:white;
      -moz-box-shadow: 0 0 1em black;/*Firefox 3.5*/
      -webkit-box-shadow: 0 0 1em black;/*Safari 3.0*/
       box-shadow: 0 0 1em black;/*CSS3*/
      }
      </style>
  </head>
  <body>
      div id="cont">
          <h3>Başlık 1</h3>
          <p>Başlık 1 içerik</p>
          <h3>Başlık 2</h3>
          <p>Başlık 2 içerik</p>
      </div>
  </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

:FIRST-OF-TYPE VE :LAST-OF-TYPE

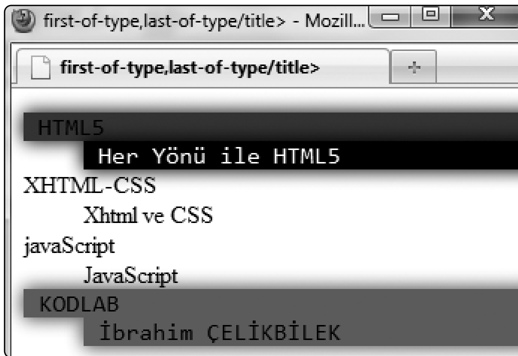
Kapsayıcı eleman içerisinde bir türün ilk çocuk elemanına CSS tanımlaması yapmak için `:first-of-type`, son çocuk elemanına CSS tanımlaması yapmak için `:last-of-type` sınıfları kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>first-of-type,last-of-type</title>
              <style type="text/css">
               dt:first-of-type{
              /*dl elemanı içerisindeki ilk dt elemanına CSS kodları uygulanır.*/
                font-family:consolas;
                background-color:blue;
                text-indent:10px;
                -moz-box-shadow: 0 0 1em blue;/*Firefox 3.5*/
                -webkit-box-shadow: 0 0 1em blue;/*Safari 3.0*/
                box-shadow: 0 0 1em blue;/*CSS3*/
              }
              dd:first-of-type{
              /*dl elemanı içerisindeki ilk dd elemanına CSS kodları uygulanır.*/
                font-family:consolas;
                background-color:black;
                text-indent:10px;
                color:white;
                -moz-box-shadow: 0 0 1em black;/*Firefox 3.5*/
                -webkit-box-shadow: 0 0 1em black;/*Safari 3.0*/
                box-shadow: 0 0 1em black;/*CSS3*/
              }
              dl *:last-of-type{
              /*dl elemanı içindeki farklı türlerin son elemanlarına CSS kodları uygulanır.*/
               font-family:consolas;
               background-color:crimson;
               text-indent:10px;
               -moz-box-shadow: 0 0 1em crimson;/*Firefox 3.5*/
               -webkit-box-shadow: 0 0 1em crimson;/*Safari 3.0*/
               box-shadow: 0 0 1em crimson;/*CSS3*/
              }
```

```
            </style>
      </head>
      <body>
         <dl>
            <dt>HTML5</dt>
            <dd>Her Yönü ile HTML5</dd>
            <dt>XHTML-CSS</dt>
            <dd>Xhtml ve CSS</dd>
            <dt>javaScript</dt>
            <dd>JavaScript</dd>
            <dt>KODLAB</dt>
            <dd>İbrahim ÇELİKBİLEK</dd>
          </dl>
      </body>
</html>
```

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

:ONLY-CHILD, :ONLY-OF-TYPE, :ROOT VE :EMPTY

:only-child kapsayıcı eleman içerisinde tek çocuk durumunda olan elemana CSS tanımlaması yapmak için kullanılır.

:only-of-type kapsayıcı eleman içerisinde bir türe ait tek çocuk durumunda olan elemana CSS tanımlaması yapmak için kullanılır.

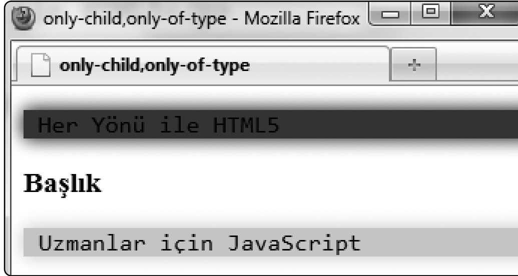
:root HTML elemanına stil tanımlaması yapmak için kullanılır.

:empty içinde bir düğüm olmayan elemanlara CSS tanımlaması yapmak için kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8"/>
            <title>only-child,only-of-type</title>
            <style type="text/css">
             div#bir p:only-child{
            /*div#bir elemanı içerisinde tek çocuk durumunda olan (Aynı ya da farklı
            türde başka bir çocuk eleman olmaması gerekir.)
            p elemanına CSS tanımlamaları uygulanır.*/
              font-family:consolas;
              background-color:blue;
              text-indent:10px;
              -moz-box-shadow: 0 0 1em blue;/*Firefox 3.5*/
              -webkit-box-shadow: 0 0 1em blue;/*Safari 3.0*/
              box-shadow: 0 0 1em blue;/*CSS3*/
            }
            div#bir+div p:only-of-type{
            /*div#bir elemanının kardeşi olan div elemanı içerisinde bir türe ait tek
             çocuk durumunda olan(Aynı türde başka bir eleman olmaması lazım.
             Farklı türde başka bir eleman olabilir) p elemanına CSS
             tanımlamaları uygulanır.*/
              font-family:consolas;
              background-color:lightblue;
              text-indent:10px;
              -moz-box-shadow: 0 0 1em lightblue;/*Firefox 3.5*/
              -webkit-box-shadow: 0 0 1em lightblue;/*Safari 3.0*/
              box-shadow: 0 0 1em lightblue;/*CSS3*/
            }
            </style>
      </head>
      <body>
        <div id="bir">
          <p>Her Yönü ile HTML5</p>
        </div>
        <div>
          <h3>Başlık</h3>
          <p>Uzmanlar için JavaScript</p>
        </div>
      </body>
</html>
```

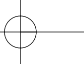
Ekran görüntüsü:

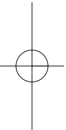


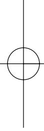
Firefox 3.6 ekran görüntüsü

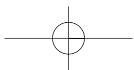
Yapisal Sözde Siniflar İçin Tarayici Desteği

<table>
<tr><th>Seçici</th><th>Internet Explorer</th><th>Firefox</th><th>Opera</th><th>Safari</th></tr>
<tr><td>:root</td><td rowspan="3">IE9</td><td>1.0+</td><td rowspan="4">9.5+</td><td>1.0+</td></tr>
<tr><td>:nth-child()
:nth-of-type()
:nth-last-child()
:nth-last-of-type()
:first-of-type
:last-of-type
:only-of-type</td><td>3.5+</td><td rowspan="3">3.1+</td></tr>
<tr><td>:last-child
:only-child
:empty</td><td rowspan="2">3+</td></tr>
<tr><td>:first-child</td><td>E8 (Kısmen)</td></tr>
</table>









HTML5 VE WEB FORMLARI

5

Kullanıcıdan bilgi almak (toplamak) için form alanları oluşturmalısınız. Kullanıcı form içerisine bilgileri girip gönder butonuna tıkladığında, form elemanları içerisindeki bilgiler sunucu taraflı çalışan bir sayfada işlenir. Sunucu taraflı çalışan sayfa, kullanıcıdan aldığı bilgilerle ilgili çeşitli işlemler yapabilir ve gerekli ise kullanıcıya geri mesaj da verebilir. Form elemanlarını kullanmak için ilk önce form etiketi kullanmanız gerekecektir.

Form alanları sadece kullanıcıdan bilgi alıp bir başka sayfaya göndermek için kullanılmaz. Bunun dışında form alanları oluşturarak formu başka sayfaya göndermeden sayfanın kendi içinde JavaScript yardımıyla çeşitli işlemler yapmanız mümkündür.

Sayfa içerisinde kullandığınız her bir form etiketi ve form elemanları için **DOM** nesneleri oluşturulur. Bu durumda oluşturduğunuz form ve form elemanları birer nesnedir. Nesne özellikleri ve metotları kullanılabilir.

HTML5 yeni form elemanları ve form elemanları için yeni özellikler tanımlar. HTML5 form yapısında; görsel veri giriş kontrolleri, girilen verinin belirli bir formatta olup olmadığını kontrol eden (istemci taraflı doğrulama sınaması yapan) yeni elemanlar barındırır.

Form Nesnesi

Form alanları oluşturmak için <form> etiketi kullanılır. Bu elemanının içerisinde form elemanları bulunur.

Metotları: checkValidity(), dispatchFormChange(), dispatchFormInput(), item(index), namedItem(name), submit(), reset()

Özellikleri: [Standart Özellikler], accept-charset, action, autocomplete, enctype, method, name, novalidate, target

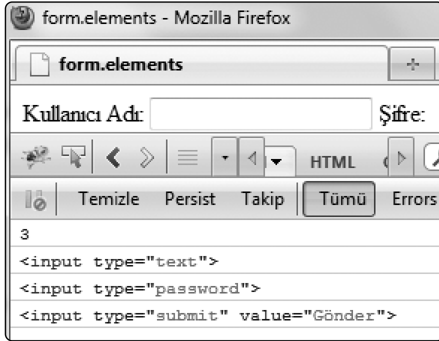
- **accept-charset:** Veri girişi için sunucu tarafından kabul edilecek karakter kodlaması listesini ayarlar.
- **action:** Formun işlenmesi için gönderileceği sayfa adı.
- **autocomplete [HTML5]:** Veri girişi yapılan form elemanları için otomatik tamamlama özelliğini aktif ya da pasif yapmak için kullanılır. Tarayıcı, kullanıcının veri alanına önceden yazdığı değerlere göre otomatik tamamlama yapar. (on ya da off değerlerinden birini alır.)
- **enctype:** Şifreleme türü.
- **method:** Gönderme metodu (get/post).
- **name:** Formun adı.
- **novalidate [HTML5]:** Form elemanları için doğrulama işleminin yapılıp yapılmayacağını ayarlar. Form etiketi içinde tanımlanan bu değer submit ve image elemanları tarafından formnovalidate özelliği ile değiştirilebilir. Boolean türü (boolean attributes) bir özelliktir. novalidate="novalidate" ya da sadece novalidate şeklinde kullanabilirsiniz.
- **target:** action ile belirtilen sayfanın açılacağı hedef pencere.
- **id:** Form için benzersiz bir kimlik.
- **length:** Form içerisindeki var olan eleman sayısı.
- **elements:** Form içerisinde kullanılmış elemanların koleksiyonunu (listesini) döndürür. Geriye HTMLFormControlsCollection tipinde bir nesne döndürür. HTMLFormControlsCollection arayüzü, form elemanları listesini saklar. Saklanan bu liste belge yapısında değişiklik olduğunda güncellenir.

Örnek:

```
<html>
       <head><meta charset="utf-8"/>
              <title>form.elements</title>
              <style type="text/css">
              </style>
       <script type="text/javascript">
              var _load=function(){
                     var frmeleman=document.forms["form_1"];
                     /*Form elemanına ulaşmak için;
                       document.form[index]
                       document.form[name]*/
                     var htmllen=frmeleman.length;
                     //Form elemanı içerisindeki eleman sayısı
                     var htmlcol=frmeleman.elements;
                     //HTMLFormControlsCollection tipinde liste
                     console.log(htmllen);
                     /*Form içerisindeki eleman sayısını yazdırdık*/
                     for(var i=0;i<htmllen;i++){
                     console.log(htmlcol[i]);
                     /*Form içerisindeki elemanları konsola yazdırmak için
                       bir döngü oluşturduk.
                       htmlcol[i]
                       htmlcol.item(index)
                       Yukarıdaki yöntemlerden herhangi birini kullanabilirdik*/
                     }
              }
       </script>
       </head>
       <body onload="_load();">
              <form action="" name="form_1">
                     Kullanıcı Adı:
                     <input type="text"/>
                     Şifre:
                     <input type="password"/>
                     <input type="submit" value="Gönder"/>
              </form>
       </body>
</html>
```

Ayrıca liste içerisindeki elemanlara ulaşmak için `collection.namedItem(name)` metodunu kullanabilirsiniz. `name` parametresi elemanın `id` ya da `name` özelliklerine aldığı değerlerden biri olabilir.

Ekran görüntüsü:



Firefox 3.6 ekran görüntüsü

Input Elemanı Özellikleri ve Type Tanımlamaları

`Input` elemanı (nesnesi), form içerisine kontroller yerleştirmek için kullanılır. `Input` nesnesinin oluşturacağı elemanın türü `type` özelliği ile tanımlanır.

Input Elemanı Özellikleri

Standart Özellikler

- `accept`
- `alt`
- `checked`
- `dirname`
- `disabled`
- `name`
- `size`
- `src`
- `type`
- `value`
- `readonly`

Aşağıdaki özellikler HTML5 ile tanımlanmıştır.

- `autocomplete`
- `autofocus`
- `form`
- `formaction`
- `formenctype`
- `formmethod`
- `formnovalidate`
- `formtarget`
- `height`
- `list`
- `max`
- `maxlength`
- `min`
- `multiple`
- `pattern`
- `placeholder`
- `required`
- `step`
- `width`

Hangi özelliğin hangi eleman için tanımlı olduğuna bakalım.

Özellikler	Input Elemanları													
	text, searh	url, tel	e-mail	password	range	color	radio checkbox	file	submit	image	reset	hidden	Datetime Date, month week, time	datetime-local, number
accept	-	-	-	-	-	-	-	*	-	-	-	-	-	-
alt	-	-	-	-	-	-	-	-	-	*	-	-	-	-
autocomplete	*	*	*	*	*	*	-	-	-	-	-	-	*	*
checked	-	-	-	-	-	-	*	-	-	-	-	-	-	-
dirname	*	-	-	-	-	-	-	-	-	-	-	-	-	-
formaction	-	-	-	-	-	-	-	-	*	*	-	-	-	-
formenctype	-	-	-	-	-	-	-	-	*	*	-	-	-	-
formmethod	-	-	-	-	-	-	-	-	*	*	-	-	-	-
formnovalida te	-	-	-	-	-	-	-	-	*	*	-	-	-	-
formtarget	-	-	-	-	-	-	-	-	*	*	-	-	-	-
height	-	-	-	-	-	-	-	-	-	*	-	-	-	-
list	*	*	*	-	*	*	-	-	-	-	-	-	*	*
max	-	-	-	-	*	-	-	-	-	-	-	-	*	*
maxlength	*	*	*	*	-	-	-	-	-	-	-	-	-	-
min	-	-	-	-	*	-	-	-	-	-	-	-	*	*
multiple	-	-	*		-	-	-	*	-	-	-	-	-	-
pattern	*	*	*	*	-	-	-	-	-	-	-	-	-	-
placeholder	*	*	*	*	-	-	-	-	-	-	-	-	-	-
readonly	*	*	*	*	-	-	-	-	-	-	-	-	*	*
required	*	*	*	*	-	-	*	*		-	-	-	*	*
size	*	*	*	*	-	-	-	-	-	-	-	-	-	-
src	-	-	-	-	-	-	-	-	-	*	-	-	-	-
step	-	-	-	-	*	-	-	-	-	-	-	-	*	*
width	-	-	-	-	-	-	-	-	-	*	-	-	-	-

Özelliklerin karşısında bulunan * karakterleri hangi elemanlar için tanımlı olduklarını gösterir.

AUTOCOMPLETE

Veri girişi yapılan form elemanlarında otomatik tamamlama özelliğini aktif ya da pasif yapmak için kullanılır. Tarayıcı, kullanıcının veri alanına önceden yazdığı değerlere göre otomatik tamamlama yapar (`on` ya da `off` değerlerinden birini alır.) Hatırlarsanız form elemanı içinde `autocomplate` özelliği bulunmaktaydı. Bir input elemanına yapacağınız `autocomplate` tanımlaması ile form elemanından kalıtsal olarak gelen `autocomplate` tanımlamasını değiştirebilirsiniz.

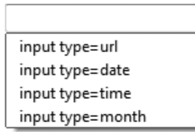
LIST

Veri girişi yapılan alanlar için önceden tanımlı bir veri listesini (seçenekler listesi) kullanıcının seçimine sunar. Kullanıcı, isterse bu seçenek listesindeki bir veriyi kullanılır ya da kendisi belirtilen alana farklı bir veri girişi yapar. Bu özelliğe aynı belge içerisinde tanımlı olan bir `datalist` elemanının `id` değeri atanır.

Örnek:

```
<input  name=”text” list=”veri”/>
   <datalist id=”veri”>
      <option value=”input type=url”/>
      <option value=”input type=date” />
      <option value=”input type=time”/>
      <option value=”input type=month”/>
   </datalist>
```

Ekran görüntüsü:



Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset=”utf-8”/>
              <title>list</title>
              <style type=”text/css”>
              </style>
       </head>
       <body>
              <input name=”color” type=”color” list=”color”/>
               <datalist id=”color”>
                      <option value=”#DC143C” label=”Crimson”></option>
                      <option value=”#DAA520” label=”GoldenRod”/>
                      <option value=”#000080” label=”Navy”/>
                      <option value=”#4169E1” label=”RoyalBlue”/>
               </datalist>
       </body>
</html>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

Dikkat ederseniz önceden tanımlı renk değerleri kullanıcıya önerilmektedir. `datalist` elemanı `HTMLDataListElement` arayüzünden oluşturulmuştur ve bu eleman içerisinde `option` elemanları ile seçenekler oluşturulur. `options` özelliğini kullanarak içerisindeki seçenek listesine programa tik olarak ulaşabiliriz.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8"/>
            <title>datalist</title>
            <style type="text/css">
                  div#result {
                        color:crimson;
                        font:consolas;
                        font-size: 14px;
                        font-style: italic;
      }
            </style>
         <script type="text/javascript">
         var goster= function() {
              var elemanLst=document.querySelector("#color");
                    //datalist Elemanının referansını aldık
              var result=document.querySelector("form#form1+div#result");
                    /*datalist içerisindeki seçenekleri yazdıracağımız
                      div#result elemanının referansını aldık*/
              var htmlCollection=elemanLst.options;
                    /*datalist Elemanı içerisindeki option
                      elemanlarının listesini aldık. options özelliği geriye
                      HTMLCollection tipinde bir nesne döndürür.*/
                    for(var i=0; i<htmlCollection.length; i++){
                    result.innerHTML +=htmlCollection.item(i).value
+ " " + htmlCollection.item(i).label+"<br/>";
                    /*for döngüsüyle HTMLCollection nesnesi (htmlCollection elemanı)
                      içerisindeki option elemanlarına ulaştık ve bu elemanların value
                      ve label özelliklerini div#result elemanı içerisine innerHTML
                      özelliğini kullanarak yazdırdık.*/
                    }
         }
</script>
```

```
        </head>
        <body>
          <form id=”form1”>
                <input name=”color” type=”color” list=”color” />
                 <datalist id=”color”>
                        <option value=”#DC143C” label=”Crimson”></option>
                        <option value=”#DAA520” label=”GoldenRod”/>
                        <option value=”#000080” label=”Navy”/>
                        <option value=”#4169E1” label=”RoyalBlue”/>
                 </datalist>
                 <input type=”button” value=”datalist seçeneklerini göster”
onclick=”goster();”/>
          </form>
          <div id=”result”>
          </div>
        </body>
</html>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

PATTERN

Veri giriş alanlarına kullanıcının belirli bir formatta (kalıpta) veri girmesini sağlamak için kullanılır. Bu özelliğe harf, rakam ve özel karakterlerden oluşturulmuş düzenli bir ifade (`RegExp`) atanır. Kullanıcının girdiği verinin bu kalıba uygun olması gerekir.

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset=”utf-8”/>
                <title>pattern</title>
        </head>
        <body>
```

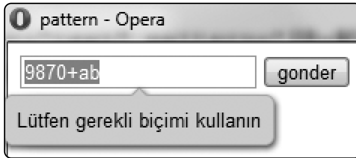
```
            <form id="form1">
            <input name="txt" type="text" pattern="[0-9]{4}-[a-z]{3}"/>
                <input type="submit" value="gonder"/>
            </form>
      </body>
</html>
```

`Pattern` özelliğine atadığımız düzenli ifadeye bakalım.

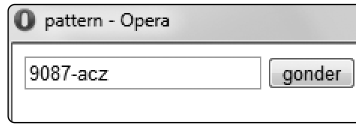
```
[0-9]{4}-[a-z]{3}
```

Bu alana `0` ile `9` arasındaki sayılardan (`0` ve `9` dahil) dört tane sonra `-` (tire) sonra `a` ile `z` harfleri arasından (`a` ve `z` dahil) üç tane harf girilebilir. Kullanıcı bu kalıba uygun bir metin yazmazsa (boş bırakabilir) ve `submit` butonuna tıklarsa, geçerlilik sınamasından dolayı uyarı mesajı ile karşılaşır.

Ekran görüntüleri:



Düzenli ifadeye uygun olmayan veri



Düzenli ifadeye uygun veri

Opera 11 ekran görüntüleri

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8"/>
            <title>pattern</title>
            <style type="text/css">
            #form1 label{
              font-family: calibri;
              color:crimson;
            }
            </style>
      </head>
      <body>
            <form id="form1">
            <label>[0262-666-66-66/İş telefonu] formatında bir
metin giriniz</label><br/>
```

```
                        <input name="txt" type="text" pattern="[0-9]{4}-
[0-9]{3}-[0-9]{2}-[0-9]{2}\D{5,}"/>
                        <input type="submit" value="gonder"/>
                </form>
        </body>
</html>
```

Düzenli ifademizi incelersek;

`[0-9]{4}-[0-9]{3}-[0-9]{2}-[0-9]{2}\D{5,}`

...`0-9` aralığında (`0` ve `9` dahil) dört sayı sonra - (tire) sonra `0-9` aralığında (`0` ve `9` dahil) üç sayı sonra `-` sonra `0-9` aralığında (`0` ve `9` dahil) iki sayı daha sonra `-` sonra `0-9` aralığında (`0` ve `9` dahil) iki sayı sonra peşinden rakam dışında kalan karakterlerden en az 5 tane kullanarak oluşturulmuş bir metin.

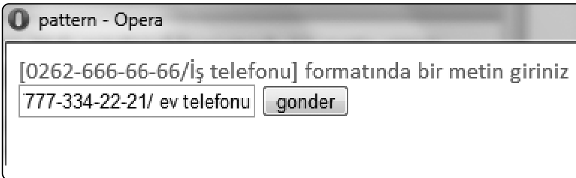
Ekran görüntüleri:

Kullanıcı tanımlanan formatta veri girişi yapmadığında:



Opera 11 ekran görüntüsü

Kullanıcı tanımlanan formatta veri girişi yaptığında;



Opera 11 ekran görüntüsü



Düzenli ifadeler (`RegExp`) ile ilgili ayrıntılı bilgi için KODLAB yayıncılıktan çıkan **JavaScript** isimli kitabımdan faydalanabilirsiniz.



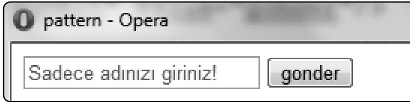
PLACEHOLDER

Bu özellik ile veri giriş alanları için yer tutucu bir değer tanımlaması yapabilirsiniz. Yani kullanıcıya belirtilen alana ne tür veri gireceği ile ilgili bir ipucu (referans) verebilirsiniz. Kullanıcı belirtilen elemana odaklandığında tanımlanan bu değer kaybolur. `value` özelliği; varsayılan bir değer tanımlaması yaparken, `placeholder` özelliği ise sadece bir ipucu tanımlaması yapar.

Örnek:

```
<form id=”form1”>
    <input name=”txt” type=”text” placeholder=”Sadece adınızı giriniz!”/>
    <input type=”submit” value=”gonder”/>
</form>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

REQUIRED

Tanımlı olduğu input elemanlarının kullanıcı tarafından doldurulmasını ya da seçim yapılmasını gerekli hale getirir. Eğer kullanıcı required özelliğinin tanımlı olduğu bir elemanı boş geçerse (veri girmez ya da seçim yapmazsa) form bilgileri gönderilirken yapılan geçerlilik denetimi dolayısıyla uyarı mesajı ile karşılaşır.

Boolean türü (b*oolean attributes*) bir özelliktir. `required=”required”` ya da sadece `required` şeklinde kullanabilirsiniz.



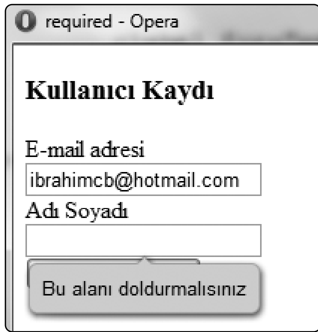
GEÇERLİLİK DENETİMİ (SINAMASI)

Kullanıcının bir denetime belirlediğimiz formatta (kalıpta), tipte ya da aralıkta veri girip girmediğinin tarayıcı tarafından kontrol edilmesidir. Geçerlilik denetimi sırasında hata oluşursa tarayıcı, form bilgilerini belirtilen adrese göndermez (istemci taraflı doğrulama işlemi).

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset=”utf-8”/>
                <title>required</title>
```

```
            <style type="text/css">
            label{
              display:block;
              width:300px;
            }
            </style>
        </head>
<body>
    <h3>Kullanıcı Kaydı</h3>
    <form action="kayit.aspx" method="post">
       <label for="email"> E-mail adresi</label>
        <input id="email" type="email" required  name="email" size="20">
       <!--size özelliği veri giriş alanlarının genişliğini ayarlamak için kullanılır-->
        <label for="text1">Adı Soyadı</label>
        <input id="text1" type="text" required name="txt" size="20"/><br/>
        <input type="submit" value="Kullanıcı Oluştur!">
    </form>
    </body>
</html>
```



Opera 11 ekran görüntüsü

Ekran görüntüsü:

Form Elemanına Ait Özellikleri Tekrar Tanımlayan Özellikler

Aşağıdaki özellikler form elemanı için tanımlanmış `action`, `enctype`, `method`, `novalidate`, `target` özelliklerini override etmek (geçersiz kılmak, üzerine yazmak) için kullanılırlar. Bu özellikler `submit` ve `image` elemanları için tanımlıdırlar.

- **formaction:** Form elemanının action özelliğini tekrar tanımlar.
- **formenctype:** Form elemanının enctype özelliğini tekrar tanımlar.
- **formmethod:** Form elemanının method özelliğini tekrar tanımlar.
- **formnovalidate:** Form elemanı için novalidate özelliğini tanımlar. Belirtilen özellik zaten tanımlıysa herhangi bir işlev yapmaz.
- **formtarget:** Form elemanının target özelliğini tekrar tanımlar.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>formnovalidate</title>
              <style type="text/css">
              p{
                margin: 5px 0px;
              }
              </style>
       </head>
       <body>
              <form action="kayit.aspx" method="post">
                   <p>Adı:</p> <input required name="ad">
                   <p>Görüşü</p> <textarea required name="gr"></textarea>
                   <p>
                   <input type="submit" name="submit1" value="Doğrulama Yap">
                   <input type="submit" formnovalidate name="submit2"
                    value="Doğrulama Yapma">
                  </p>
              </form>
       </body>
</html>
```

Eğer kullanıcı **Doğrulama Yap** butonuna tıklarsa form gönderilmeden doğrulama işlemi yapılır. **Doğrulama Yapma** butonuna tıklanırsa doğrulama işlemi yapılmaz.

Ekran görüntüsü:

Opera 11 ekran görüntüsü

TYPE TANIMLAMALARI

Type özelliği; Input etiketi ile oluşturulacak elemanın türünü ayarlamak için kullanılır. Aşağıda bu özelliğe atanabilcek değerler listelenmiştir.

BUTTON

Bu form elemanı ile bir olayı tetikleyip JavaScript kodlarının çalışmasını sağlayabilirsiniz.

CHECKBOX

Onay kutusu ile kullanıcının birçok seçenek içerisinden istediği kadar seçim yapabileceği eleman grupları oluşturabilirsiniz. checked özelliği ile checkbox elemanının seçili olma durumunu öğrenebilir ya da ayarlayabilirsiniz

PASSWORD

Kullanıcının girdiği verileri maskeleyen metin kutusudur. Formlarda sıkça kullanacağınız bir form elemanıdır. Metin kutusuna girdiğiniz veriler * karakteri ile gösterilirler.

RADIO

Seçenek düğmesi ile kullanıcının bir grup seçenek içerisinden sadece bir seçim yapması isteniliyorsa, bir radio eleman grubu oluşturabilirsiniz. checked özelliği ile radio elemanının seçili olma durumunu öğrenebilir ya da ayarlayabilirsiniz.

IMAGE

Submit butonu yerine kullanabileceğiniz bir resim butonu oluşturur.

SUBMIT

Forma girilen verileri action özelliği ile belirtilen sayfaya gönderir. Sunucu taraflı çalışan sayfa kullanıcıdan aldığı bilgilerle ilgili çeşitli işlemleri yerine getirir.

TEXT

Kullanıcının bilgi gireceği bir metin kutusu oluşturur. `type` özelliği atanmazsa ya da yanlış yazılırsa otomatik olarak metin kutusu oluşur.

RESET

Kullanıcı tarafından forma girilen verileri siler, yani formu temizler.

FILE

Dosyalarınızı sunucudaki herhangi bir konuma göndermek için kullanılır. `multiple` özelliği ile birden fazla dosya seçimi yapılabilir.

HIDDEN

Bu eleman formda gösterilmez. Elemanın içerisinde `name` özelliğine bir değer atarsanız bir de `value` değeri tanımlarsınız gönder butonuna tıkladığınızda `value` özelliğine atadığınız değer `name` özelliği ile tanımladığınız değişkene atanmış olur.

EMAIL [HTML5]

Kullanıcının sadece e-posta adresi girebileceği bir form elemanı oluşturur. Submit butonuna tıklandığında ya da `submit()` metodu çağrıldığında Bu alana girilen veriye geçerlilik denetimi yapılır.

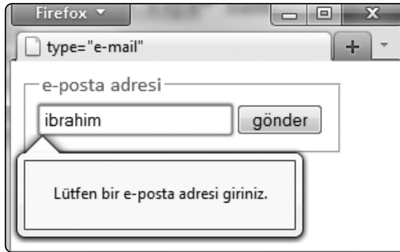
Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>type="e-mail"</title>
              <style type="text/css">
              form{
                width:250px;
                height: 130px;
              }
              legend{
                font-family:calibri;
                color:crimson;
              }
```

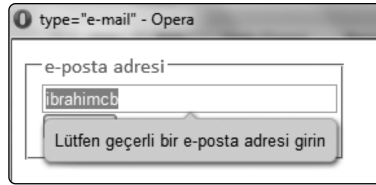
```
                </style>
        </head>
        <body>
                <form action="isle.aspx" name="form1">
                <fieldset>
                <legend>e-posta adresi</legend>
                <input name="email" type="email" required/>
                <input type="submit" value="gönder" />
                </fieldset>
                </form>
        </body>
</html>
```

`<input type="email" required/>` ile `required` özelliği kullanılarak kullanıcının bu alana veri girmesi gerekli hale getirilmiştir. Kullanıcı bu alanı doldurmadan ya da geçerli bir e-posta adresi yazmadan submit butonuna tıklarsa geçerlilik sınamasından dolayı uyarı mesajı ile karşılaşır.

Ekran görüntüleri:



Firefox 4.0 ekran görüntüsü



Opera 11 ekran görüntüsü

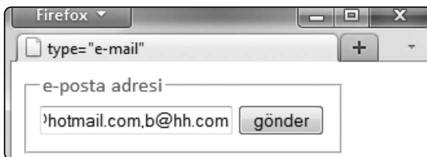
`multiple` özelliğini kullanarak arada virgül olmak kaydıyla birden fazla e-mail adresi girilebilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset="utf-8"/>
                <title>type="e-mail"</title>
                <style type="text/css">
                form {
                  width:250px;
```

```
            height:130px;
          }
          legend {
            font-family:calibri;
            color:crimson;
          }
          </style>
    </head>
    <body>
          <form action="islem.aspx" name="form1">
          <fieldset>
          <legend>e-posta adresi</legend>
          <input name="email" type="email" multiple list="liste"/>
           <datalist id="liste">
                 <option value="zeynep@hotmail.com"></option>
                 <option value="alis@hotmail.com" />
                 <option value="yigit@ornek.com"/>
                 <option value="asmin@ornek.com"/>
           </datalist>
          <input type="submit" value="gönder" />
          </fieldset>
          </form>
    </body>
</html>
```

Ekran görüntüsü:



Firefox 4.0 ekran görüntüsü

SEARCH, URL VE TEL [HTML5]

- **search:** Aranılacak metin ya da metin gruplarını yazmak için kullanılır. Bu elemana tek satırlık metin grupları yazılabilir.
- **url:** Bir URL adresi girmek için kullanılır. Varsayılan olarak form bilgileri gönderilirken geçerlilik sınaması (doğrulama kontrolü) yapılır.
- **tel:** Telefon numarası girmek için tanımlanır.

Yukarıdaki input elemanlarında pattern özelliği kullanılarak kullanıcının belirli bir formatta bilgi girmesi sağlanabilir. Ya da maxlength özelliği ile kullanıcının girebileceği en uzun veri ayarlanabilir.

NUMBER [HTML5]

Sayısal değerler yazmak için kullanılır. min özelliği ile girilebilecek en düşük değer max özelliği ile girilebilecek en yüksek sayısal değer ayarlanabilir. step özelliği ise; min, max değerleri arasında, bu alana girilebilecek diğer sayıları tanımlamak için artış miktarını ayarlar. Elemana girilen değere value özelliği ile programa tik olarak ulaşabilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>number</title>
       </head>
       <body>
          <form action="kayit.aspx" method="post">
              <input type="number" min="10" max="100" step="10"
name="num"/>
              <input type="submit" />
          </form>
       </body>
</html>
```

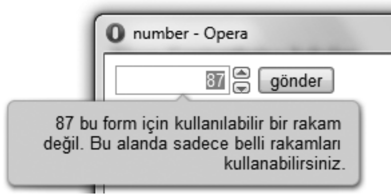
Ekran görüntülerine bakalım...

Eğer kullanıcı min ve max değerleri arasında bir değer girmezse:



Opera 11 ekran görüntüsü

Kullanıcı min, max değerleri arasında, fakat step özelliği ile belirtilen artış miktarına uygun bir sayı girmezse:



Opera 11 ekran görüntüsü

Kullanıcı; `min`, `max` değerleri arasında ve `step` özelliği ile belirtilen artış miktarına uygun bir sayı girerse herhangi bir hata oluşmayacaktır.



Opera 11 ekran görüntüsü

RANGE [HTML5]

Kaydırma çubuğu kontrolü; `min` ve `max` özellikleri ile tanımlanan aralıkta görsel olarak (tutamaç ile) bir değer elde etmek için kullanılır. `min` özelliği ile aralığın en düşük değeri, `max` özelliği ile aralığın en yüksek sayısal değeri ayarlanır. `step` özelliği ise `min`, `max` değerleri arasında tutamaç ile elde edilebilecek diğer sayıları tanımlamak için artış miktarını ayarlar. Bu eleman ile üretilen değere `value` özelliği ile ulaşılır.

Varsayılan değerler:

- `min:0`
- `max:100`
- `step:1`
- `value: min + (max-min)/2`

Örnek: `range`, elemanının sakladığı (ürettiği) değeri `text` elemanı içerisinde gösterelim.

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>type="range"</title>
<script type="text/javascript">
var init=function(){
  change();
  /*sayfa yüklendiğinde kaydırma çubuğunun ürettiği değeri göstermek için change()
  Fonksiyonunu çalıştıralım*/
}
```

```
var change=function(){
//formchange olayında çalışacak fonksiyon
/*form elemanının sakladığı değerde (ya da görsel olarak üretilen değerde) değişiklik
olursa formchange olayı oluşur.*/
  var elSlider=document.querySelector("input[type='range']");
  var elResult=document.querySelector("input[type='range']+input");
  elResult.value=elSlider.value;
}
</script>
       </head>
       <body onload="init();">
              <form action="" name="form1">
              <input type="range" name="rng" max="100" min="20"
step="5" onformchange="change();" />
              <input type="text" size="5"/>
              </form>
       </body>
</html>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

COLOR [HTML5]

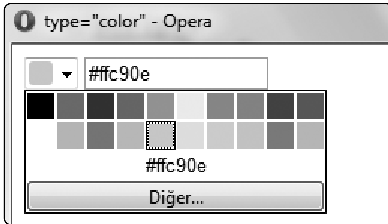
Renk seçimi yapmak (*color picker - renk seçici ile*), renk değeri girmek için kullanılan elemandır. Girilen renk değerine value özelliği ile ulaşabilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8"/>
              <title>type="color"</title>
<script type="text/javascript">
var change=function(){
  var elColor=document.querySelector("input[type='color']");
  var elResult=document.querySelector("input[type='color']+input");
  elResult.value=elColor.value;
```

```
}
</script>
        </head>
        <body>
                <form action="" name="form1">
                <input type="color" name="clr"  onformchange="change();" />
                <input type="text" size="15" />
                </form>
        </body>
</html>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

Kullanıcı `color picker` ile görsel olarak ya da rengin hexadecimal değerini yazarak renk seçimi yapabilir. `Color` elemanının `value` özelliğinde seçilen rengin hexadecimal formatındaki karşılığı saklanır.

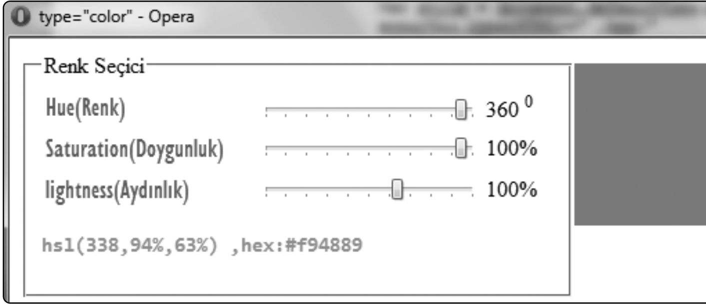
Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8"/>
            <title>type="color"</title>
            <style type="text/css">
                  form {
                    width:400px;
                    position: relative;
                  }
                  form tr td:nth-of-type(1) {
                    width:150px;
                    color:crimson;
                    font-family:"Gill Sans MT Condensed";
                    font-size:20px;
```

```
        }
        div#renk {
          width:100px;
          height:118px;
          position:absolute;
          right:-100px;
          top:10px;
        }
        p#result{
          font-family: consolas;
          font-size: 15px;
          color:#11b0ee;
          font-weight: bold;
        }
  </style>
<script type="text/javascript">
    var ch= function() {
        var hsl = document.querySelector("input[name='hue']");
        var saturation =
        document.querySelector("input[name='saturation']");
        var light=document.querySelector("input[name='light']");
        /*range elemanlarının referanslarını aldık*/
        var target=document.getElementById("renk");
        /*div#renk şeklinde tanımlı elemanın referansını aldık.
        var colorStr="hsl("+hsl.value+"," + saturation.value +
        "%," +light.value+ "%)";
        /*Kaydırma çubukları ile üretilen sayısal değerleri hsl formatında bir
        renk değerine dönüştürdük ve colorStr değişkenine atadık*/
        target.style.backgroundColor=colorStr;
        /*Üretilen bu renk değerini div#renk elemanına
        arkaplan rengi olarak atadık.*/
        var sonucYaz=document.getElementById("result");
        sonucYaz.innerHTML=colorStr;
        /*Üretilen hsl formatındaki değeri p#result elemanının içine yazdırdık*/
        var style =
        document.defaultView.getComputedStyle(target, null);
        sonucYaz.innerHTML += " ,hex:" +
        style.getPropertyValue("background-color");
        /*div#renk elemanının arkaplan rengine ulaştık ve p#result elemanı
        içerisine yazdırdık*/
  }
```

```
            </script>
      </head>
      <body>
            <form action="" name="form1">
                  <fieldset>
                    <legend>
                    Renk Seçici
                    </legend>
                    <table>
                        <tr>
                            <td>Hue(Renk)</td>
                            <td>
                            <input type="range" name="hue" min="0"
max="360" step="1" onformchange="ch();"/>360<sup>0</sup></td>
                        </tr>
                        <tr>
                            <td>Saturation(Doygunluk)</td>
                            <td>
                            <input type="range" name="saturation" min="0"
max="100" step="1" onformchange="ch();"/>100%</td>
                        </tr>
                        <tr>
                            <td>lightness(Aydınlık)</td>
                            <td>
                  <input type="range" name="light" min="0" max="100"
step="1" onformchange="ch();"/>100%</td>
                        </tr>
                    </table>
                    <p id="result">
                    </p>
                  </fieldset>
                  <div id="renk">
                  </div>
            </form>
      </body>
</html>
```

Ekran görüntüsü:



Opera 11 ekran görüntüsü

Kaydırma çubuklarını hareket ettirdiğimizde, renk seçicinin hemen yanındaki div elemanında üretilen rengi görebiliyoruz. Ayrıca üretilen rengin hsl ve hex formatlarındaki karşılıkları da hemen altta yazmakta. Zaten rengin hsl formatındaki değerini kaydırma çubuklarından üretilen değerlerle oluşturduğumuzdan, bunu biliyoruz. Peki, rengin hexadecimal karşılığını nasıl elde ettik diye sorabilirsiniz. Biz, div elemanının arkaplan rengini hsl formatında tanımladık, fakat `getComputedStyle` metodunu kullanarak `div#renk` elemanının arkaplan rengini tekrar elde ettik. Çünkü bu metot geriye rengin Firefox, Safari (RGB formatında), Opera (`Hexadecimal` formatında) karşılığını döndürecektir. `getComputedStyle()` metoduna biraz daha yakından bakalım...

Bir HTML elemanı için tanımlanan CSS özelliklerine atanan değerlere ulaşmak ve değiştirmek mümkün mü? Bu soruya şöyle cevap verelim.

NOT

STYLE ÖZELLİĞİ

```
<ELEMENT style = "CSS Bildirimleri" >   // style Attribute
object.style [ = "CSS Bildirimi" ]   // style  Property
```

- `style` (attribute) özelliği ile (*In-Line-Satır İçi*) tanımlanan CSS özelliklerin aldıkları değerlere ulaşmak ve değiştirmek mümkündür. Ayrıca yine satır içi olacak şekilde yeni CSS özellikleri tanımlayabilirsiniz.

  ```
  <p id="elP" style="color:red;">Satır içi (In-line CSS Tanımlaması)</p>
  ```

  Elemanı nesne olarak elde edelim.

```
var elemanP=document.getElementById("elP");
console.log(elemanP.style.color);        // Konsolda "red" yazar
```

```
elemanP.style.color="blue";
// p elemanının yazı rengini değiştirdik.
```

Dikkat ederseniz elemanı nesne olarak elde ettikten sonra `style` (property) özelliği kullanılarak **`color`** CSS özelliğine ulaşıldı ve konsola yazdırıldı. Elemanının `style` (attribute) özelliğinde tanımlanmamış bir CSS özelliğine `style` (property) ile ulaşamayız. `style` (property) özelliği geriye `CSSStyleDeclaration` tipinde bir nesne döndürür.

Eğer `style` (property) özelliğini kullanarak bir eleman için satır içi (in-line) CSS tanımlaması yapmak istersek ya da satır içi tanımlanan bir CSS özelliğinin değerini değiştirmek istersek şu şekilde kullanabiliriz.

```
elemanReferansı.style.CSSözellikAdı="değer";
```

Eğer elemana satır içi olarak tanımlanmış bir CSS özelliğinin değerine ulaşmak istersek şu şekilde kullanabiliriz.

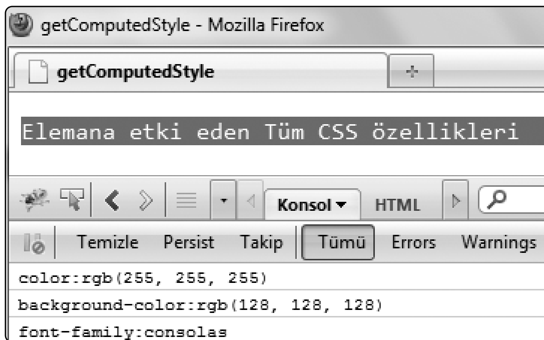
```
elemanReferansı.style.CSSözellikAdı
```

- Bir HTML elemanını hedef alan (HTML elemanına uygulanan) farklı alanlarda tanımlanan (`style` (attribute) özelliği ile `<style>` etiketleri arasında, `<link>` etiketi ile, tarayıcı tarafından varsayılan olarak tanımlanan) tüm CSS özelliklerine ulaşmak için `getComputedStyle()` metodu kullanılır. Bu metot, geriye `CSSStyleDeclaration` tipinde sadece okunabilir bir CSS özellik listesi döndürür.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8"/>
            <title>getComputedStyle</title>
            <style type="text/css">
        p:nth-of-type(1){
          /*body elemanı içindeki ilk p elemanı hedef alındı*/
          color:blue;
          background-color: gray;
          font-family: consolas;
        }
            </style>
            <script type="text/javascript">
var play= function() {
```

```
  var elemanP=document.getElementById("elP");
  var computedStyle = document.defaultView.getComputedStyle(elemanP, null);
  console.log("color:"+computedStyle.getPropertyValue("color"));
  console.log("background-color:" +
computedStyle.getPropertyValue("background-color"));
  console.log("font-family:" +
computedStyle.getPropertyValue("font-family"));
}
             </script>
       </head>
       <body onload="play();">
       <p id="elP" style="color:white;">Elemana etki eden Tüm CSS
özellikleri</p>
  </body>
</html>
```



Firefox 4.0 ekran görüntüsü

Dikkat ederseniz renk değerlerini renk ismi şeklinde tanımlamış olmama rağmen `getComputedStyle()` metodu renk değerlerini geriye rgb formatında (**Firefox*, *Safari*) döndürdü.

DATE, MONTH, WEEK, TIME, DATETIME-LOCAL, DATETIME [HTML5]

- **date:** Görsel olarak tarih değeri girilebilecek form elemanı oluşturur. (*Yıl/Ay/Gün*)
- **month:** Görsel olarak ay ve yıl değeri girilebilecek form elemanı oluşturur. (*Yıl/Ay)*
- **week:** Görsel olarak hafta ve yıl değeri girilebilecek form elemanı oluşturur. (*Yıl/Hafta*)

Hafta bilgisi tanımlanırken; [W+Yıl içindeki kaçıcı hafta] şeklinde bir format kullanılır. Örneğin; W01: Yılın ilk haftası.

- **time:** Saat ve dakika değeri girilebilecek form elemanı oluşturur.
- **datetime-local:** Tarih (*Yıl/Ay/Gün*) ve saat (*saat:dakika*) değerleri girilebilecek form elemanı oluşturur.
- **datetime:** Tarih (*Yıl/Ay/Gün*) ve saat (*saat:dakika*) değerleri girilebilecek form elemanı oluşturur. (UTC zaman standardına göre.)

Bu elemanlar ile üretilen tarih ya da saat bilgisine; elemanların `value` (*property*) özelliğini kullanarak erişebilirsiniz.

Opera 11 ile ekran görüntüleri:

`<input type="date" name="date" />`

Date Pickers - Opera

2011-03-08

Mart 2011

Pzt	Sal	Çrş	Prş	Cum	Cts	Paz
28	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Bugün

`<input type="month" name="month" />`

Date Pickers - Opera

2011-03

Mart 2011

Pzt	Sal	Çrş	Prş	Cum	Cts	Paz
28	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Bugün

`<input type="week" name="week" />`

Date Pickers - Opera

2011-W10

Mart 2011

Hafta	Pzt	Sal	Çrş	Prş	Cum	Cts	Paz
9	28	1	2	3	4	5	6
10	7	8	9	10	11	12	13
11	14	15	16	17	18	19	20
12	21	22	23	24	25	26	27
13	28	29	30	31	1	2	3
14	4	5	6	7	8	9	10

Bugün

`<input type="time" name="time" />`

Date Pickers - Opera

01:09

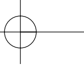
`<input type="datetime-local" name="dtlocal" />`

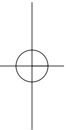
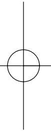
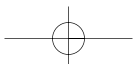
Date Pickers - Opera

2011-03-09 12:09



`type` özelliğinin yeni değerleri için tarayıcı desteği ilk bölümde anlatılmıştır.



6 CANVAS

HTML5 dilinin getirdiği en önemli elemanlardan biri hiç şüphesiz <canvas> elemanıdır. Bu elaman bir çizim alanı tanımlar ve bu çizim alanında javaScript yardımıyla grafikler oluşturulur ya da düzenlenir. **Canvas** (*tuval*) elemanını kullanarak vektörel grafikler, oyun grafikleri hatta animasyonlar oluşturabilir ya da resimlerinizi piksel tabanlı olarak düzenleyebilirsiniz. Bu eleman ile görüntüler dinamik olarak piksel tabanlı oluşturulur. Bu eleman javaScript ile dinamik grafikler oluşturmak ya da var olan resimleri piksel tabanlı olarak düzenlemek için kullanıcıya güçlü bir imkan sunar. Ayrıca sunucu tarafındaki yükün azaltılması için bu görüntülerin istemci tarafında oluşturulması da önemlidir. Bu eleman **HTMLCanvasElement DOM** arayüzü ile tanımlıdır.

Özellikleri: `[Standart Özellikler], width, height`

`width` özelliği ile `canvas` elemanının genişliği, `height` özelliği ile canvas elemanının yüksekliği ayarlanır. Aşağıda basit olarak bir canvas elemanı oluşturalım.

```
<canvas id="canvas" width="300px" height="100px">
      Tarayıcı canvas elemanını desteklemiyor...
</canvas>
```

`canvas` etiketleri arasına tarayıcının desteklememesi olasılığına karşın bir metin yazabilirsiniz. Eğer tarayıcı bu elemanı desteklemezse içerisindeki metni görüntüleyecektir. Eğer `width` ve `height` özelliklerine değer atamazsanız varsayılan olarak `width=300px`, `height=150px` değerlerini alır.

Metotları: `getContext(), toDataUrl()`

getContext()

Bu metot canvas elemanı ile tanımlanan alanı belirtilen boyutta bir çizim alanına dönüştürür. Bu metot geriye grafik oluşturmak için gerekli özellik ve metotlara sahip `CanvasRenderingContext2D` tipinde bir nesne döndürür. `getContext()` metodu ile oluşturulan bu çizim alanının varsayılan olarak arkaplan rengi şeffaf siyahtır.

Kullanımı:

```
CanvasRenderingContext2D Nesnesi =
canvasElemanReferansı.getContext(contextId)
```

`contextId` parametresi ile çizim alanının kaç boyutlu olacağı ayarlanır. Bu parametreye yerine **2D** (*2 boyutlu çizim alanı*) değeri ya da çok fazla desteklenmemekle beraber **webgl** değeri yazılabilir. **webgl** değeri standart olmamakla beraber `WebGLRenderingContext` tipinde 3 boyutlu bir çizim alanı nesnesi oluşturur.

`getContext()` metodunun kullanımıyla ilgili basit bir örnekle işe başlayalım.

```
<!DOCTYPE html>
<html>
      <head><meta charset=”utf-8”/>
            <title>canvas</title>
            <script type=”text/javascript”>
      function draw() {
            var canvas = document.getElementById(“canvas”);
            if(canvas.getContext){
              // Eğer tarayıcı getContext() metodunu destekliyorsa çalışacak kodlar
              var context2d = canvas.getContext(“2d”);
              // context2d değişkeni CanvasRenderingContext2D tipinde bir nesnedir.
              context2d.fillStyle = “rgb(200,0,0)”;
              context2d.fillRect (10, 10, 55, 50);
            }
      }
            </script>
      </head>
      <body onload=”draw();”>
            <canvas id=”canvas” width=”300” height=”100”>
                  Tarayıcı canvas elemanını desteklemiyor...
            </canvas>
      </body>
</html>
```

Canvas elemanının yerleşimine ve içerisindeki çizime görsel olarak bakalım.



toDataUrl()

canvas elemanı içerisindeki çizimi (resmi, grafiği) tanımlanan (Örneğin; PNG) formatta temsil eden bir resim verisi döndürür.

Kullanımı: `ImageData = canvasElemanReferansı.toDataURL(ImageType)`

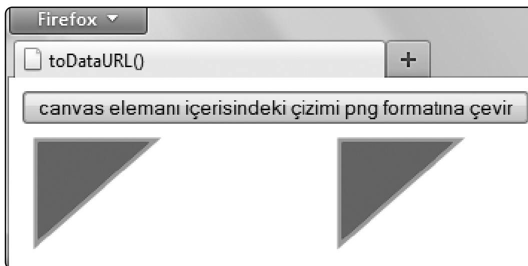
Bu metot içerisine canvas içerisindeki resim bilgisinin hangi formatta elde edileceği yazılır. *image/png* değeri şu an için tarayıcılar tarafından ortak olarak desteklenen tek format olarak karşımıza çıkmaktadır. Canvas elemanı içerisindeki resim bilgisini bu metotla öğrenip bir <img> elemanına resim kaynağı olarak tanımlayabilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8"/>
            <title>toDataURL()</title>
            <script type="text/javascript">
      function draw() {
            var canvas = document.getElementById("canvas");
            var context2d = canvas.getContext("2d");
            context2d.fillStyle = '#E32756';
            context2d.strokeStyle = '#3BC429';
            context2d.lineWidth = 3;
            context2d.beginPath();
            context2d.moveTo(10, 10);
            context2d.lineTo(90, 10);
            context2d.lineTo(10, 80);
            context2d.closePath();
            context2d.fill();
            context2d.stroke();
```

```
        }
        var copyImage= function() {
                var canvas = document.getElementById("canvas");
                var img=document.getElementById("img");
                var dataImage=canvas.toDataURL("image/png");
                // canvas içerisindeki resim bilgisini png formatında aldık
                img.src=dataImage;
                // img referanslı resim elemanın src özelliğine atadık.
        }
                </script>
        </head>
        <body onload="draw();">
                <div>
                        <button onclick="copyImage()">
                        canvas elemanı içerisindeki çizimi png formatına çevir
                        </button>
                </div>
                <div>
                        <canvas id="canvas" width="200" height="100" >
                        </canvas>
                        <img id="img">
                </div>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Canvas RenderingContext2D (2 Boyutlu Çizim Alanı) Nesnesi Özellik ve Metotları (Çizim Oluşturmak)

Canvas elemanı üzerinde çizim yapmak için <canvas> elemanı ile tanımlanan alanı getContext("2d") metoduyla bir çizim alanına dönüştürmeniz gerekir. getContext("2d") metodu 2 boyutlu çizimler oluşturmak için geriye Canvas RenderingContext2D tipinde bir nesne döndürür. İşte şimdi bu nesnenin özellik ve metotlarına bakalım. Bu bölümde anlatım boyunca Canvas RenderingContext2D nesnesi kısaca context2d olarak adlandırılacaktır.

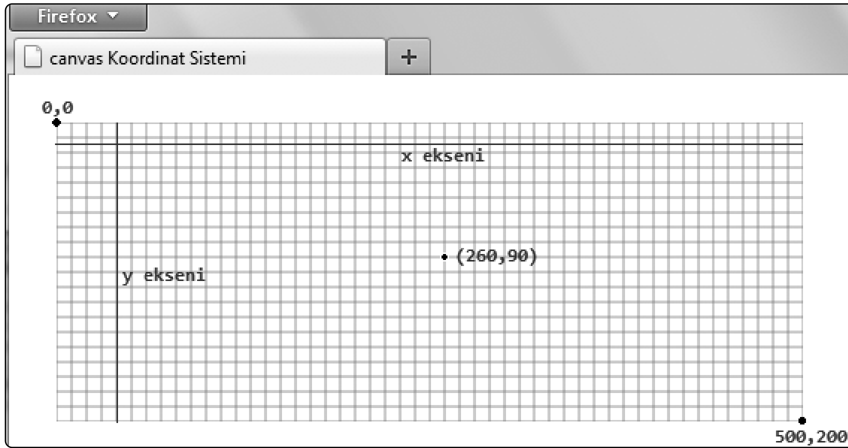
Canvas Koordinat Sistemi

Canvas elemanının büyüklüğü width, height özellikleri ile tanımlanır. Bu eleman içerisinde 2 boyutlu çizimler yaparken varsayılan olarak orijin noktası sol üst köşe olarak kabul edilir. Çizimlerde kullandığımız ölçü birimi (uzunluk birimi) px'dir. Aşağıdaki örnek ile canvas elemanının koordinat sistemine görsel olarak bakalım.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>canvas Koordinat Sistemi</title>
        <style type="text/css">
            canvas {
                margin: 20px;
            }
        </style>
        <script type="text/javascript">
            var init = function(){
                var canvasEl = document.querySelector("canvas");
                var ctx = canvasEl.getContext("2d");
                // canvasRenderingContext2d nesnesini elde ettik
                ctx.strokeStyle = "#00ccff";
                for (var i = 0; i <= canvasEl.width; i += 10) {
                    /*Varsayılan olarak çizime başlangıç noktası canvas(0,0) dır.
                      moveTo() metoduyla çizim işlemi için yeni bir
                      başlangıç noktası tanımlanır.*/
```

```
                         ctx.moveTo(i, 0);
                         ctx.lineTo(i, 200);
                        /*lineTo() metodu belirtilen koordinatlar  arasında bir
                         çizgi çizer.*/
                    }
                    for (var i = 0; i <= canvasEl.height; i += 10) {
                        ctx.moveTo(0, i);
                        ctx.lineTo(500, i);
                    }
                    ctx.stroke();
                    // Çizgi oluştur
                }
            </script>
        </head>
        <body onload="init();">
            <canvas id="canvas" width="500" height="200">
                tarayıcı canvas elemanını desteklemiyor.
            </canvas>
        </body>
</html>
```

Ekran görüntüsüne anlaşılması için kılavuzlar ekleyelim ve sonuca bakalım.



Firefox 4 ekran görüntüsü

CONTEXT2D NESNESİ ÖZELLİKLERİ

`context2D` nesnesi için tanımlı olan özellikleri inceleyelim.

CANVAS

Canvas elemanının referansını almak için kullanılır. Sadece okunabilir (*readonly*) bir özelliktir. Bu özellik sayesinde canvas elemanının `width`, `height` özelliklerine tekrar ulaşabilirsiniz.

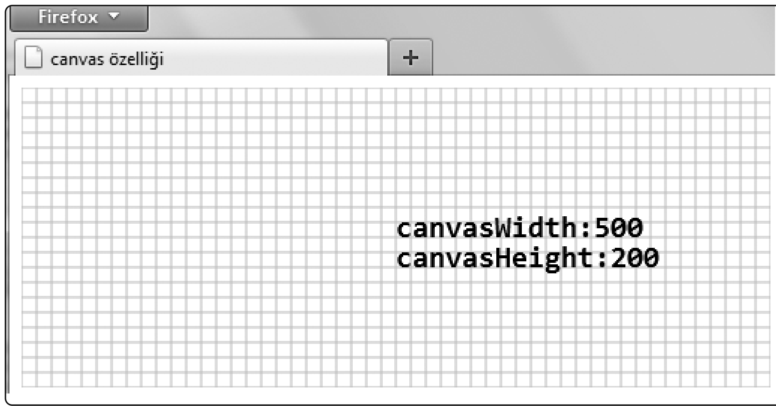
Kullanımı: `context2D.canvas`

Örnek:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>canvas özelliği</title>
        <script type="text/javascript">
            var init = function(){
                var canvasEl = document.querySelector("canvas");
                var ctx = canvasEl.getContext("2d");
                ctx.strokeStyle = "#c8c8c8";
                for (var i = 0; i <= canvasEl.width; i += 10) {
                    ctx.moveTo(i, 0);
                    ctx.lineTo(i, 200);
                }
                for (var i = 0; i <= canvasEl.height; i += 10) {
                    ctx.moveTo(0, i);
                    ctx.lineTo(500, i);
                }
                ctx.stroke();
                ctx.font = "bold 20px consolas";
               /*2 boyutlu çizim alanına dönüştürdüğümüz canvas elemanının
                yükseklik ve genişlik değerlerine ulaşalım;*/
                var canvasW = "canvasWidth:" + ctx.canvas.width;
                var canvasH = "canvasHeight:" + ctx.canvas.height;
               /*Belirtilen değerleri yatayda orta noktadan başlamak kaydıyla
                canvas elemanı içerisine yazdıralım*/
                ctx.fillText(canvasW, ctx.canvas.width / 2, 100);
                ctx.fillText(canvasH, ctx.canvas.width / 2, 120);
            }
```

```
        </script>
    </head>
    <body onload="init();">
        <canvas id="canvas" width="500" height="200">
            tarayıcı canvas elemanını desteklemiyor.
        </canvas>
    </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

strokeStyle ve fillStyle Özellikleri

Vektörel şekil (grafik/çizim) için dolgu rengi `fillStyle` özelliği ile, kenarlık rengi ise `strokeStyle` özelliği ile tanımlanır ya da önceden yapılan tanımlamalar, bu özelliklerle elde edilir. Bu özelliklerin varsayılan renk değerleri siyahtır.

Kullanımları:

```
context2D.fillStyle[= value]
context2D.strokeStyle[= value]
```

Bu özelliklere CSS renk değeri veya `CanvasGradient`, `CanvasPattern` nesnelerinden birinin adı yazılabilir. Örneğin; aşağıdaki tanımlamaların hepsinde `fillStyle` özelliğine mavi rengi atanmıştır.

```
context2D.fillStyle="blue";
context2D.fillStyle="rgb(0,0,255)";
```

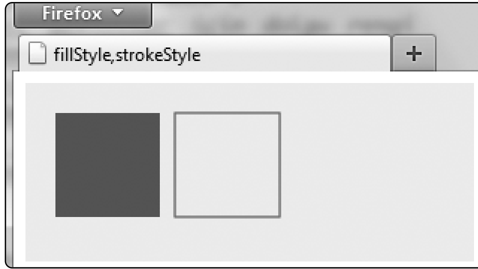
```
context2D.fillStyle="rgba(0,0,255,1)";
context2D.fillStyle="#0000FF";
```

Dikkat ederseniz renk tanımlaması yaparken `rgba()` formatını kullanarak bir saydamlık değeri tanımlayabiliyoruz.

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset="utf-8">
                <title>fillStyle,stroke style</title>
                <style type="text/css">
                        canvas {
                                background-color: #efefef;
                        }
                </style>
                <script type="text/javascript">
        var draw= function() {
                var canvasEl=document.querySelector("#canvas");
                var ctx=canvasEl.getContext("2d");
                // ctx , canvasRenderingContext2d tipinde bir nesnedir
                ctx.fillStyle="#127A19";
                // Vektörel şekilller için dolgu rengi
                ctx.strokeStyle="rgb(255,9,23)";
                // Vektörel şekiller için kenarlık rengi
                ctx.fillRect(20,20,70,70);
                // Sadece dolgu alanı olan dikdörtgen(fillStyle özelliğini kullanır)
                ctx.strokeRect(100,20,70,70);
                // Sadece kenarlığı olan bir dikdörtgen(strokeStyle özelliğini kullanır)
        }
                </script>
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="300" height="120">
                        tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Ekran görüntüsü:

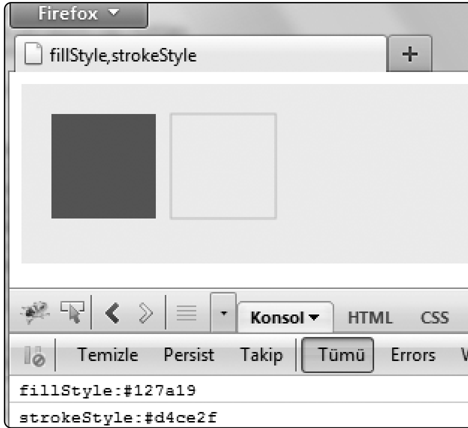


Firefox 4 ekran görüntüsü

`fillStyle`, `strokeStyle` özellikleri ile tanımlanan renk değerleri kendilerinden sonra gelen ve çizim işlemi yapan metotlar için geçerli olurlar (Serbest çizim alanları oluştururken dolgu ve kenarlık renkleri `fill()`, `stroke()` metotlarıyla uygulanır. Bu durumda `fillStyle`, `strokeStyle` özellikleri `fill()`, `stroke()` metotlarından önce tanımlanmalıdırlar). Context2D çizim formatında bu özelliklerin kod sıralamasındaki tanımlandığı yer önemlidir. Daha iyi anlaşılması için aşağıdaki örneği inceleyiniz.

```
var draw= function() {
      var canvasEl=document.querySelector("#canvas");
      var ctx=canvasEl.getContext("2d");
            ctx.fillStyle="#127A19";
            /*Yukarıdaki stil tanımlaması aşağıda tanımlanan fillRect()
              metodu tarafından kullanılır*/
            ctx.strokeStyle="rgb(255,9,23)";
            /*İlk tanımlanan strokeRect() metodu için geçersiz*/
            console.log("fillStyle:"+ctx.fillStyle);
            // Geçerli olan fillStyle tanımlaması
            ctx.fillRect(20,20,70,70);
            ctx.strokeStyle="#D4CE2F";
            /*Aşağıda tanımlanan strokeRect() metodu tarafından kullanılır*/
            ctx.fillStyle="rgb(0,255,0)";
            /*İlk tanımlanan fillRect() metodu için geçersiz*/
            console.log("strokeStyle:"+ctx.strokeStyle);
            // Geçerli olan strokeStyle tanımlaması
            ctx.strokeRect(100,20,70,70);
            /*Aşağıdaki stil tanımlamarı yukarıdaki dikdörtgenlere uygulanmaz*/
            ctx.fillStyle="#411180";
            ctx.strokeStyle="#E820CA";
}
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Yukarıdaki örnekten de anlaşılacağı üzere `fillRect()`, `strokeRect()` metotlarının kullanacağı stil tanımlaması kod akışında kendilerinden önce yapılan ilk tanımlamadır. Diğer bir yandan benzer olmakla beraber serbest çizim alanları oluştururken `fill()`, `stroke()` metotları ile beraber kullanımları ileride anlatılacaktır.

GLOBALALPHA

`context2D` nesnesi üzerinde oluşturduğunuz vektörel şekil ya da resimlere saydamlık (şeffaflık) tanımlaması yapmak ya da daha önce yapılan saydamlık tanımlamasını öğrenmek için `globalAlpha` özelliği kullanılır. Bu özelliğe `0.0` ve `1.0` arasında bir değer atanabilir. `0` değeri tam saydam, `1` değeri hiç saydam değil anlamına gelir. Varsayılan değeri `1.0`'dır.

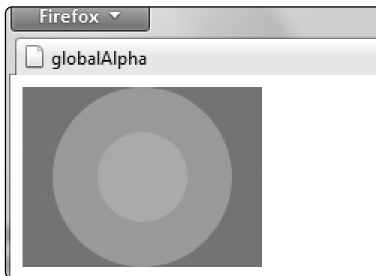
Kullanımı: `context2D.globalAlpha[= value]`

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>globalAlpha</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              // Stil tanımlamaları
```

```
            ctx.fillStyle="#FC3A3A";
            // Dikdörtgen için geçerli olacak stil tanımlaması
            ctx.fillRect(0,0,160,120);
            // fillRect ile dikdörtgen çizdik.
            ctx.fillStyle="#ffffff";
            /*Yukarıdaki stil tanımlaması arc() metodunda kullanılmak üzere tanımlandı*/
            for(var i=1;i<3;i++) {
                   ctx.globalAlpha=0.2*i;
                   /*globalAlpha özelliği ilk daire için 0.2
                   ikinci daire için 0.4 olacak*/
                   ctx.beginPath();
                   ctx.arc(80,60,i*30,0,2*Math.PI,true);
                   ctx.fill();
                   // Daire çizmek için tanımlandılar.
            }
      }
            </script>
      </head>
      <body onload="draw();">
            <canvas id="canvas" width="160" height="120">
                   Tarayıcı canvas elemanını desteklemiyor.
            </canvas>
      </body>
</html>
```

Ekran Görüntüsü:



Firefox 4 ekran görüntüsü

`globalAlpha` özelliği; `fillStyle`, `strokeStyle` özellikleri ile benzer olarak tanımlandığı satırdan sonra çizdirilen vektörel şekil ya da resimlere uygulanır.

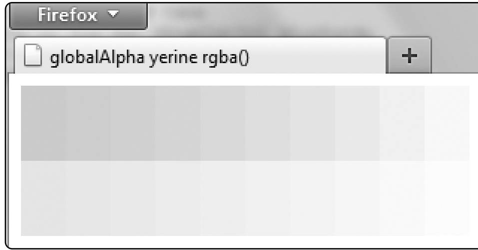
`globalAlpha` özelliği; vektörel şeklin ya da resmin tamamı için bir saydamlık değe-

ri tanımlar Yani tanımlandığı satırdan sonra çizim işlemi yapan tüm metotların kullanacağı bir saydamlık değeri tanımlar. İsterseniz `fillStyle`, `strokeStyle` özelliklerine `rgba()` formatında renk tanımlayarak vektörel şekillerin kenarlık ya da dolguları için ayrı ayrı saydamlık tanımlaması yapabilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>globalAlpha</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.fillStyle ='#3DF540';
              ctx.fillRect(0,0,300,50);
              ctx.fillStyle ='#E2F53D';
              ctx.fillRect(0,50,300,50);
              var i=0;
              // Şeffaflık değerini değiştirmek için i değişkeni tanımladık
              for(var x=0;x<=270;x+=30) {
                     /*Beyaz renkli fakat şeffaflığı değişen 20 tane dikdörtgen
                       çizeceğiz bunun için iç içe for döngülerini oluşturdu
                       çizeceğimiz bu dikdörtgenlerin width:30,height:50 olacak.*/
                     ctx.fillStyle="rgba(255,255,255,"+ i/10 +")";
                     i++;
                     for(var y=0;y<=50;y+=50) {
                            ctx.fillRect(x,y,30,50);
                     }
              }
       }
              </script>
       </head>
       <body onload="draw();">
              <canvas id="canvas" width="300" height="200">
                     Tarayıcı canvas elemanını desteklemiyor.
              </canvas>
       </body>
</html>
```

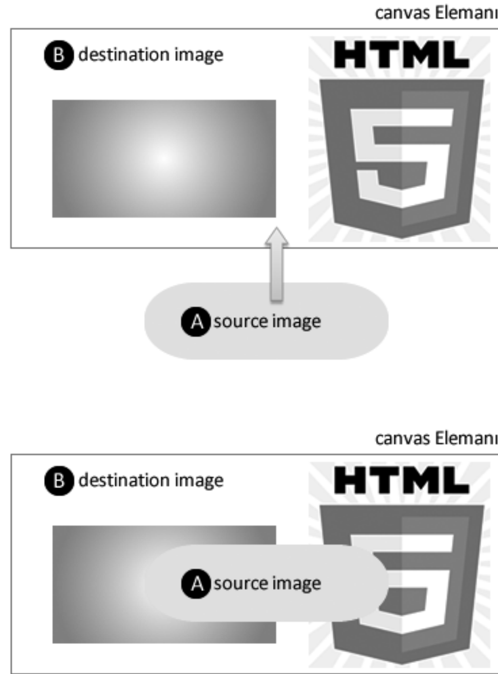
Ekran görüntüsü:



Firefox 4 ekran görüntüsü

globalCompositeOperation

Canvas üzerine bir grafik (vektörel şekil ya da resim) çizdirdiğinizde varsayılan olarak bu grafik; belirlediğiniz konuma ve canvas üzerinde bulunan diğer grafiklerin üzerine çizilecektir (Tüm canvas alanı bir resmi temsil eder). Bu durumda yeni grafik ile önceden canvas üzerinde bulunan grafikler üst üste binebilir, yani kesişme alanları oluşabilir. İşte tam bu noktada canvas üzerine yeni bir grafik eklemek istediğinizde tarayıcının gösterdiği varsayılan olan bu davranışı değiştirmek için `globalCompositeOperation` özelliği kullanılabilir. Bu özellik hem okunabilir hem de yazılabilir bir özelliktir. Yandaki şemaları inceleyelim.



Canvas üzerinde var olan grafiklerin (çizimler, resimler) tamamı bir resmi oluşturur. Yukarıdaki gösterimde; canvas üzerinde bulunan grafik **B-destination image** (*hedef resim*), canvas üzerine çizdirilecek yeni grafik **A-source image** (*kaynak resim*) şeklinde tanımlanmıştır.

Kullanımı: `context2D.globalCompositeOperation[ = value ]`

Bu özelliğe aşağıdaki değerlerden biri atanabilir.

copy, destination-atop, destination-in, destination-out, destination-over, lighter, source-atop, source-in, source-out, source-over(varsayılan değer), xor

globalCompositeOperation özelliği; yukarıda anlatılan diğer özellikler gibi tanımlandığı satırdan sonra çizdirilen vektörel şekil ya da resimlerin canvas'a nasıl yerleştirileceğini (grafiklerin çizim modunu) ayarlar.

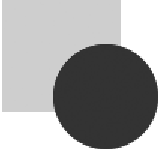
Aşağıda örnekle bu değerlerin nasıl davranışları temsil ettiklerine bakalım.

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>globalCompositeOperation</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var context2d=canvasEl.getContext("2d");
              context2d.fillStyle="#1BFA39";
              context2d.fillRect(10,10,80,80);
              context2d.fillStyle="blue";
              // globalCompositeOperation="değer"
              // özelliği bu satırda  tanımlanacak...
              context2d.beginPath();
              context2d.arc(80,80,35,0,2*Math.PI,true);
              context2d.fill();
       }
              </script>
       </head>
       <body onload="draw();">
              <canvas id="canvas" width="200" height="150">
                     Tarayıcı canvas elemanını desteklemiyor.
              </canvas>
       </body>
</html>
```

globalCompositeOperation özelliğine değerleri tek tek atayarak sonuçlara bakalım.

Aşağıdaki gösterimde canvas üzerinde olan resim kare (*hedef resim B*) yeni eklenecek grafik ise daire (*kaynak resim A*) şeklinde kullanılmıştır. canvas alanının tamamı hedef resmi temsil eder. Hedef resim içinde birden fazla resim ya da şekil olabilir. Biz aşağıdaki örnekte hedef resim olarak sadece bir kare kullandık.

```
context2D.globalCompositeOperation="source-over"
```



Varsayılan olan bu davranışta Kaynak Resim(A), Hedef resim(B)'nin üzerine çizilir.

```
context2D.globalCompositeOperation="source-in"
```



Kaynak Resim(A)'nın Hedef Resim(B) ile kesişen alanı gösterilir.

```
context2D.globalCompositeOperation="source-out"
```



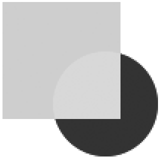
Kaynak Resim(A)'nın Hedef Resim(B) ile kesişmeyen alanı gösterilir.

```
context2D.globalCompositeOperation="source-atop"
```



Hedef Resim (B ile beraber Kaynak Resim(A)'nın Hedef Resim(B) ile kesişen alanı çizilir.

```
context2D.globalCompositeOperation="lighter"
```



Kaynak Resim(A)ile Hedef resim(B)'nin kesiştiği alanın rengi, Kesişen alanlardaki renklerin belirli oranda toplamıyla elde edilir.

```
context2D.globalCompositeOperation="xor"
```



Kaynak Resim(A)ile Hedef resim(B)'nın kesiştiği alanın rengi saydam (şeffaf) olarak çizilir. Kesişen alanda A ya da B resimleri gösterilmez, fakat kesişen alan saydamdır.

context2D.globalCompositeOperation=”destination-over”



Hedef Resim(B) üstte gösterilir.

context2D.globalCompositeOperation=”destination-in”



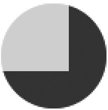
Hedef Resim (B)’nin Kaynak Resim(A) ile kesişen alanı gösterilir.

context2D.globalCompositeOperation=”destination-out”



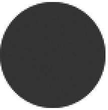
Hedef Resim(B)’nin Kaynak Resim(A) ile kesişmeyen alanı gösterilir.

context2D.globalCompositeOperation=”destination-atop”



Kaynak Resim (A) ile beraber Hedef Resim(B)’nin
Kaynak Resim(A) ile kesişen alanı çizilir.

context2D.globalCompositeOperation=”copy”



Hedef Resim(B) Komple silinir ve sadece Kaynak Resim(A) gösterilir.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset=”utf-8”>
              <title>globalCompositeOperation</title>
              <script type=”text/javascript”>
       var draw= function() {
              var canvasEl=document.querySelector(“#canvas”);
              var context2d=canvasEl.getContext(“2d”);
              var image=document.createElement(“img”);
              image.src=”zeynep.png”;
              image.onload= function() {
                     context2d.drawImage(image,0,0,170,155);
```

```
                    // resmi canvas üzerine çizdik
                    context2d.globalCompositeOperation="destination-in";
                    /*Bu satırdan sonra çizilecek grafikler için çizim modunu
                      ayarladık. Resmin (zeynep.png) daire ile kesişen
                      alanı gösterilecek */
                    context2d.beginPath();
                    context2d.arc(76,87,43,0,2*Math.PI,true);
                    // Daire çizidik
                    context2d.fill();
            }
        }
            </script>
        </head>
        <body onload="draw();">
            <canvas id="canvas" width="170" height="155">
                tarayıcı canvas elemanını desteklemiyor.
            </canvas>
        </body>
</html>
```

Resmin orijinal hali ve tarayıcı ekran görüntüsü:



zeynep.png



Firefox 4 ekran görüntüsü

Yukarıdaki örnekten de anlaşılacağı üzere maskeleme işlemleri için `globalCompositeOperation` özelliği kullanılabilir.

shadowColor, shadowOffsetX, shadowOffsetY, shadowBlur

Bir grafiğe (çizim, metin, resim) gölge rengi tanımlamak için `shadowColor` özelliği kullanılır. Bu özelliğe CSS renk değerleri atanır. Bu özellik geriye tanımlanan CSS renk değerini döndürür.

Kullanımı: `context2d.shadowColor[=value]`

Grafik ile gölge arasındaki yataydaki uzaklığı ayarlamak için `shadowOffsetX`, dikeydeki uzaklığı ayarlamak için `shadowOffsetY` özelliği kullanılır. Bu özellikler ile gölgenin pozisyonu ayarlanır. Bu özelliklerin varsayılan değeri `0`'dır. Pozitif ya da negatif bir değer kullanılabilir.

Kullanımları:

```
context2d.shadowOffsetX[=value]
context2d.shadowOffsetY[=value]
```

Gölge için uygulanacak bulanıklık miktarını ayarlamak için `shadowBlur` özelliği kullanılır. Bu özelliğin varsayılan değeri `0`'dır.

Kullanımı: `context.shadowBlur[=value]`

Yukarıdaki özeklikler hem okunabilir hem yazılabilir özellikleridir (*read/write*). Yukarıda anlatılan diğer özellikler gibi tanımlandıkları satırdan sonra çizdirilen grafikler için uygulanırlar.

Gölge; aslında grafiğin belirtilen stilde, konumda ve tanımlanan bulanıklık değerine göre oluşturulmuş bir kopyasıdır.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8">
            <title>shadow</title>
            <script type="text/javascript">
      var draw= function() {
            var canvasEl=document.querySelector("#canvas");
            var context2d=canvasEl.getContext("2d");
            // Stil tanımlamaları
            context2d.fillStyle="black";
            /*Dolgu rengi tanımlaması(fillText metodu da bu rengi kullanacak)*/
            context2d.shadowColor="#F2FF63";
            // Gölge için renk
            context2d.shadowOffsetX=6;
            context2d.shadowOffsetY=-6;
            /*Grafik ile gölge arasındaki yatay ve dikey uzaklıklar ayarlandı*/
```

```
                context2d.shadowBlur=3;
                // Gölge için bulanıklık değeri tanımlandı
                context2d.fillRect(20,20,150,80);
                /*Dikdörtgen çizdirildi sonra yukarı da anlatılan işlemler metin için
                   farklı değerlerle tekrarlandı*/
                context2d.shadowColor="#441FCC";
                context2d.shadowOffsetX=1;
                context2d.shadowOffsetY=-1;
                context2d.shadowBlur=1;
                context2d.font="18px calibri";
                // Yazı boyutu ve font ayarlandı
                context2d.fillText("canvas uygulamaları",20,120);
                // Metin ekrana yazdırıldı
        }
                </script>
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="200" height="155">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

lineWidth, lineCap, lineJoin ve miterLimit

Serbest çizim alanlarında kullanılacak ya da kenarlık çizen metotların kullanacağı çizgi kalınlığını `lineWidth` özelliği ile ayarlıyoruz.

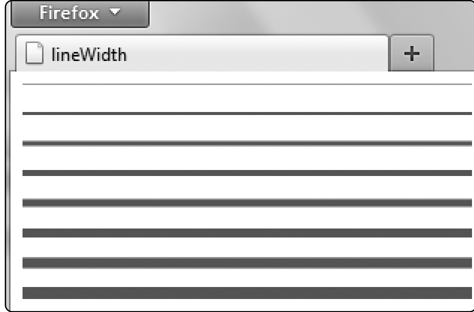
Kullanımı: `context2d.lineWidth[=value]`

Bu özelliğin başlangıç değeri `1.0`'dır. Bu özelliğe pozitif sayı değerleri atanabilir.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>lineWidth</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var context2d=canvasEl.getContext("2d");
              context2d.strokeStyle="#AA23DB";
              var i=0;
              /*i değişkenini çizgi kalınlığını değiştirmek ve çizgilerin alt alta
                çizilmesi için yeni y değerleri üretmek için kullanacağız*/
              while(i<10) {
                     context2d.lineWidth=i+1;
                     context2d.beginPath();
                     context2d.moveTo(0,i*20);
                     context2d.lineTo(300,i*20);
                     context2d.stroke();
                     i++;
              }
       }
              </script>
       </head>
       <body onload="draw();">
              <canvas id="canvas" width="300" height="155">
                     tarayıcı canvas elemanını desteklemiyor.
              </canvas>
       </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Serbest çizim alanları oluşturulurken çizilen çizgilerin sonlarının (çizgi uçlarının) stilini ayarlamak içi `lineCap` özelliği kullanılır.

Kullanımı: `context2d.lineCap[=value]`

Bu özelliğe `butt(varsayılan*)`, `round`, `square` değerlerinden biri atanabilir.

- **`butt:`** Varsayılan değerdir. Çizgi belirtilen koordinatlar arasına çizilir.
- **`round:`** Çizgi uçlarına bir yarım daire eklenir. Eklenen dairenin yarıçapı çizgi genişliğinin (`lineWidth`) yarısıdır.
- **`square:`** Çizgi uçlarına çizgi kalınlığının (`lineWidth`) yarısı kadar genişliğe sahip olan dikdörtgenler ekler.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>lineCap</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var context2d=canvasEl.getContext("2d");
              context2d.strokeStyle="#AA23DB";
              /*Kenarlıklar için stil tanımlaması yapıldı*/
              context2d.lineWidth=20;
              /*Kenarlıklar için kalınlık tanımlaması yaptık ve aşağı da 3 tane
                çizgi çizdireceğiz*/
```

```
            context2d.beginPath();
            /*İlk çizgimiz için herhangi bir lineCap tanımlaması yapmadığımızdan
            varsayılan lineCap  değeri olan 'butt' değerini kullanacak*/
            context2d.moveTo(30,20);
            context2d.lineTo(200,20);
            /*Tanımlanan koordinatlara çizgimizi stroke()
            metoduyla çizdirelim*/
            context2d.stroke();
            context2d.beginPath();
            context2d.lineCap="square";
            context2d.moveTo(30,60);
            context2d.lineTo(200,60);
            context2d.stroke();
            context2d.beginPath();
            context2d.lineCap="round";
            context2d.moveTo(30,100);
            context2d.lineTo(200,100);
            context2d.stroke();
        }
            </script>
        </head>
        <body onload="draw();">
            <canvas id="canvas" width="300" height="175">
                tarayıcı canvas elemanını desteklemiyor.
            </canvas>
        </body>
</html>
```

Ekran görüntüsüne bakalım:



Firefox 4 ekran görüntüsü

`lineCap` özeliği; `round`, `square` değerleri ile beraber kullanıldığında çizgi sonlarına çizgi kalınlığının yarısı genişliğinde bir yarım daire ya da dikdörtgen ekliyor. Bu durumu hassas çizimler yaparken dikkate almayı unutmayınız!

Diğer bir özelliğimiz ise `lineJoin`'dir. Bu özellik çizgi birleşim noktalarının nasıl gösterileceğini ayarlar.

Kullanımı: `context.lineJoin[=value]`

Bu özelliğe `bevel`, `round`, `miter` değerlerinden biri atanabilir. Varsayılan (önceden tanımlı) değer `miter`'dir.

Örnek:

```
<!DOCTYPE html>
<html>
	<head><meta charset="utf-8">
		<title>lineCap</title>
		<script type="text/javascript">
	var draw= function() {
		var canvasEl=document.querySelector("#canvas");
		var context2d=canvasEl.getContext("2d");
		context2d.lineWidth=15;
		// lineCap özelliğini buarada tanımlayacağız.
		context2d.moveTo(30,70);
		context2d.lineTo(80,15);
		context2d.lineTo(120,70);
		context2d.lineTo(170,15);
		context2d.stroke();
	}
		</script>
	</head>
	<body onload="draw();">
		<canvas id="canvas" width="300" height="100">
			tarayıcı canvas elemanını desteklemiyor.
		</canvas>
	</body>
</html>
```

`lineCap` özelliğe değerleri atayarak sonuca bakalım.

Varsayılan değer; `miter`:



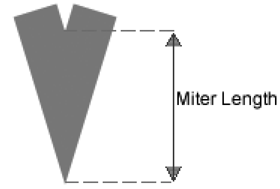
`bevel` değeri:



`round` değeri:



Bu özellik gurubundaki son özelliğimiz; `miterLimit` özelliğidir. Çizgiler eğer dar açı ile birleşmiş ise birleşim alanlarında iç ve dış kesişim noktaları oluşacaktır. Bu iki kesişim noktaları arasındaki uzaklık **miter length** (*açı ölçer uzaklığı*) olarak adlandırılır. İşte bu özellik, oluşabilecek maksimum **miterLength** değerini tanımlamak için kullanılır (Bu özellik, bir oran tanımlar). Bu özelliğin varsayılan değeri `10` tamsayısıdır.



Kullanımı: `context.miterLimit[=value]`

Yukarıda anlatılan özellikler hem okunabilir hem de yazılabilir özelliklerdir. Kullanım formatındaki `[=value]` tanımı bunu ifade etmektedir. Bu özellikler tanımlandıkları satırdan sonra çizdirilen grafikler için uygulanırlar.

CONTEXT2D NESNESİ METOTLARI

`context2D` nesnesi için tanımlı olan metodları inceleyelim.

SAVE() VE RESTORE()

Canvas üzerinde çizim yaparken kesinlikle çok işinize yarayacak metotlardır. `save()` metodu tanımlandığı satırda geçerli olan stil özelliklerinin aldıkları değerleri, transformasyon matrisi durumunu ve `clip()` metoduyla oluşturulan kırpılmış alanı daha sonra tekrar kullanılmak için hafızada saklar/kaydeder. Program akışında `save()` metoduyla saklanan bu değerleri tekrar kullanmak isterseniz; `restore()` metodunu kul-

lanabilirsiniz. `restore()` metodu kullanıldığı satırda daha önce `save()` metoduyla tanımlanan tüm stil özelliklerini (`stil+transformasyon matrisi+clip()`) aldıkları değerlerle beraber tekrar geçerli kılar.

`save()` metoduyla saklanan değer çeşitleri:

- Transformasyon matrisi,
- `clip()` metoduyla tanımlanan kırpılmış alan,
- Aşağıdaki stil özelliklerinin değerlerini saklar.

Özellikler:

- strokeStyle
- fillStyle
- globalAlpha
- lineWidth
- lineCap
- lineJoin
- miterLimit
- shadowOffsetX
- shadowBlur
- shadowColor
- globalCompositeOperation
- font
- textAlign
- textBaseline

Kullanımları:

```
context2d.save()
context2d.restore()
```

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>save(),restore()</title>
    <style>
    canvas
    {
        border:1px solid gray;
        }
    </style>
    <script type="text/javascript">
        var draw = function () {
            var ctx =
document.getElementsByTagName ("canvas")[0].getContext("2d");
```

```
            ctx.fillStyle = "#18189E";
            ctx.shadowOffsetX = 6;
            ctx.shadowOffsetY = -6;
            ctx.shadowColor = "#F725AE";
            ctx.save();
            ctx.fillRect(20, 20, 60, 60);
            ctx.fillStyle = "red";
            ctx.shadowOffsetX =3;
            ctx.shadowOffsetY =-3;
            ctx.shadowColor = "#2ABAF7";
            ctx.fillRect(100,20, 60, 60);
            ctx.restore();
            /*Bu noktadan sonra geçerli olacak stil özellikleri save() metoduyla
                saklanan stil özellikleridir. Bunlar;
                ctx.fillStyle = "#18189E";
                ctx.shadowOffsetX = 6;
                ctx.shadowOffsetY = -6;
                ctx.shadowColor = "#F725AE";
            */
            ctx.fillRect(180,20,60,60);
        }
    </script>
</head>
<body onload="draw();">
    <canvas id="canvas" width="300" height="150">
        Tarayıcı canvas elemanını desteklemiyor.
    </canvas>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Dikdörtgen Çizim Metotları

Canvas elemanı üzerine sadece dolgu alanına ya da kenarlığa sahip dikdörtgenler çizdirmek ya da canvas üzerinde dikdörtgen olarak tanımlanan bir alanının temizlenmesini sağlamak için kullanılırlar.

fillRect()

Bu metot; tanımlanan koordinatlarda sadece dolgu alanına sahip bir dikdörtgen çizmek için kullanılır. Bu metot, `fillStyle` özelliği ile tanımlanan rengi kullanır. Bu metot çizim işleminde `canvas(0,0)` noktasını (canvas sol üst köşesi) referans alır.

Kullanımı: `context2d.fillRect(x, y, w, h)`

`x`, `y` parametreleri `canvas(0,0)[canvas sol üst köşesi]` orijin noktası ile tanımlanan koordinat sisteminde yeni çizilecek dikdörtgenin sol üst köşesinin koordinatlarını tanımlamak için kullanılır (`x` parametresi ile yataydaki uzaklık, `y` parametresi ile dikeydeki uzaklık ayarlanır). `w`, `h` parametreleri çizilecek dikdörtgenin genişliğini ve yüksekliğini tanımlar.

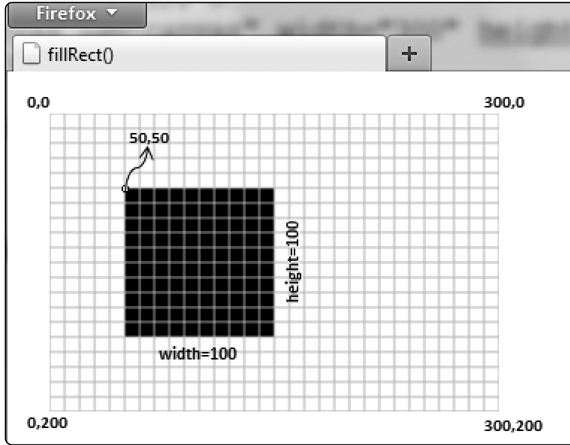
`Context2d` metotları, uzunluk birimi olarak `px` kullanır.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8">
            <title>fillRect()</title>
            <script type="text/javascript">
```

```
        var draw= function() {
                var canvasEl=document.querySelector("#canvas");
                var ctx=canvasEl.getContext("2d");
                ctx.strokeStyle="#F0B4C5";
                ctx.fillRect(50,50,100,100);
                /*Aşağıdaki döngüler ve içerisinde kullanılan metotlar canvas üzerinde
                fillRect() metodunun daha iyi anlaşılması için bir ızgara oluşturmak için
                kullanıldı. İlerleyen sayfalarda aşağıdaki metotların ne işe yaradıklarını
                daha iyi anlayacaksınız.*/
                for(var x=0;x<=300;x+=10) {
                // Dikey çizgiler için
                        ctx.beginPath();
                        ctx.moveTo(x,0);
                        ctx.lineTo(x,200);
                        ctx.stroke();
                }
                for(var y=0;y<=200;y+=10) {
                // Yatay çizgiler için
                        ctx.beginPath();
                        ctx.moveTo(0,y);
                        ctx.lineTo(300,y);
                        ctx.stroke();
                }
        }
                </script>
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="300" height="200"
style="margin:20px;">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Görsel yerleşime bakalım...



strokeRect()

Bu metot tanımlanan koordinatlarda sadece kenarlığa sahip bir dikdörtgen çizmek için kullanılır. Bu metodun kullanacağı kenarlık rengi `strokeStyle` özelliği ile tanımlanır. `fillRect()` metodunun kullandığı yerleşim formatı, bu metot içinde geçerlidir.

Kullanımı: `context.strokeRect(x, y, w, h)`

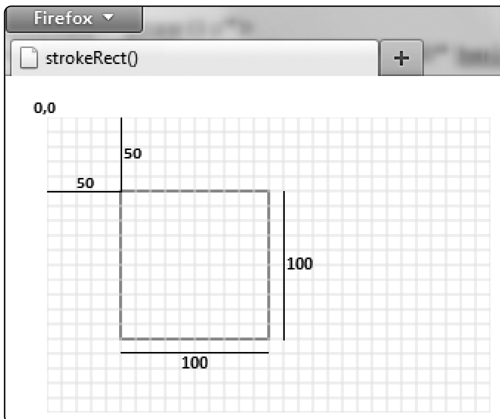
`x`, `y`, `w`, `h` parametreleri `fillRect()` metodu parametreleri ile aynı görevi görür.

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8">
            <title>strokeRect()</title>
            <script type="text/javascript">
      var draw= function() {
            var canvasEl=document.querySelector("#canvas");
            var ctx=canvasEl.getContext("2d");
            ctx.strokeStyle="red";
            ctx.lineWidth=2;
            ctx.strokeRect(50,50,100,100);
            /*Aşağıdaki döngüler ve içerisinde kullanılan metotlar canvas üzerinde
            strokeRect() metodunun daha iyi anlaşılması için bir ızgara oluşturur.
            İlerleyen sayfalarda aşağıdaki metotları ne işe yaradıklarını daha iyi
            anlayacaksınız!*/
```

```
                ctx.strokeStyle="#E6E6E6";
                ctx.lineWidth=1;
                for(var x=0;x<=300;x+=10) {
                // Dikey çizgiler için
                        ctx.beginPath();
                        ctx.moveTo(x,0);
                        ctx.lineTo(x,200);
                        ctx.stroke();
                }
                for(var y=0;y<=200;y+=10) {
                // Yatay çizgiler için
                        ctx.beginPath();
                        ctx.moveTo(0,y);
                        ctx.lineTo(300,y);
                        ctx.stroke();
                }
        }
                </script>
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="300" height="200"
style="margin:20px;">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Görsel yerleşime bakalım...



Canvas'ın uzunluk birim olarak px kullandığını unutmayınız. Yeni çizilen ve sadece kenarlık alanına sahip olan dörtgen `100 px` genişliğinde ve yüksekliğinde, bu dörtgenin sol üst köşesinin koordinatları (`x=50px`, `y=50px`) şeklindedir.

clearRect()

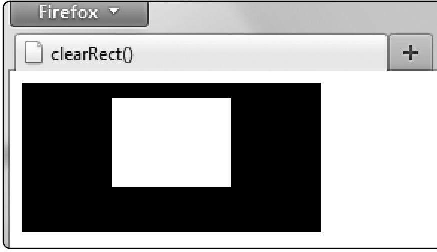
Bu metot şeffaf siyah renkli dolguya sahip bir dikdörtgen çizer. Özetle çizildiği alandaki piksellerin renk değerlerini temizler. `fillRect()`, `strokeRect()` metotlarının kullandığı yerleşim formatı bu metot içinde geçerlidir.

Kullanımı: `context.clearRect(x, y, w, h)`

`x`, `y`, `w`, `h` parametreleri `fillRect()` metodu parametreleri ile aynı görevi görür.

Örnek:

```
<!DOCTYPE html>
<html>
	<head><meta charset="utf-8">
		<title>clearRect()</title>
		<script type="text/javascript">
	var draw= function() {
		var canvasEl=document.querySelector("#canvas");
		var ctx=canvasEl.getContext("2d");
		ctx.fillRect(0,0,200,100);
		/*Siyah renkli bir dolgu alanı oluşturuldu.
		fillRect metodu fillStyle özelliği ile tanımlanan rengi kullanır.
		fillStyle özelliği tanımlanmadığından bu özelliğin varsayılan değeri olan
		siyah metot tarafından kullanılacaktır.*/
		ctx.clearRect(60,10,80,60);
		/*Seffaf siyah dolgulu yani renk değerleri temizlenmiş bir alan oluşturuldu*/
	}
		</script>
	</head>
	<body onload="draw();">
		<canvas id="canvas" width="200" height="100">
			Tarayıcı canvas elemanını desteklemiyor.
		</canvas>
	</body>
</html>
```



Firefox 4 ekran görüntüsü

Path Metotları (Karmaşık Şekiller Çizmek)

Canvas üzerine özel şekiller çizmek için path metotları kullanılır. Bu metotlar bize serbest çizim alanları oluşturma imkanı sağlar. Koordinat sisteminde tanımlayacağınız iki nokta arasında düz ya da kıvrımlı çizgiler çizdirebilir ya da bu çizgileri kullanarak bir geometrik şekil elde edebilirsiniz. Oluşturulan bu geometrik şekle bir dolgu rengi tanımlayabilirsiniz. Bu metotları kullanarak dikdörtgen türü geometrik şekillerde oluşturmamız mümkündür. Şimdi bu metotların neler olduğuna bakalım.

beginPath()

Yeni bir serbest çizime başlamak için kullanılır. Aslında yaptığı işlem çizim işleminde en son kullanılan ve yeni bir çizim için başlangıç noktası olarak (referans olarak) kabul edilecek çizim noktası bilgisini iptal etmektir. Bu metot çağrıldıktan sonra yeni çizim işlemi için bir başlangıç noktası tanımı kalmaz. Bu metottan sonra çizime başlangıç noktası (çizim noktası olarak adlandıracağız) tanımlamak için `moveTo()` metodu kullanılır. Kendisinden sonra tanımlanmış olan yol listesine `stroke()`, `fill()` metotlarının uygulanmasını sağlar. Ayrıca `closePath()` metoduyla beraber kapalı çizim alanları oluşturmak için kullanılır.

Kullanımı: `context2d.beginPath()`

moveTo()

`beginPath()` metodu çağrıldıktan sonra çizim işlemine başlamak için bir çizim noktası tanımlamak ya da serbest çizim alanı oluştururken yeni bir çizim noktası tanımlamak için kullanılır. Bu metot ile tanımlanan nokta çizim işlemi için başlangıç noktasını temsil eder.

Eğer `beginPath()` metodunu çağırarak yeni bir çizim oluşturacaksanız `moveTo()` metoduyla bir başlangıç çizim noktası oluşturmanız şarttır.

Kullanımı: `context2d.moveTo(x,y)`

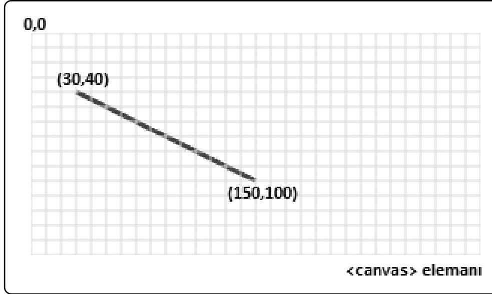
x, y değerleri koordinat sistemindeki çizim noktasını tanımlar.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>moveTo()</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.strokeStyle="#1548ED";
              ctx.lineWidth=3;
              ctx.beginPath();
              ctx.moveTo(30,40);
              ctx.lineTo(150,100);
              ctx.stroke();
              /*Aşağıdaki kodlar canvas üzerindeki ızgarayı oluşturmak için kullanıldı*/
              ctx.strokeStyle="#A9F5BB";
              ctx.lineWidth=1;
              for(var y=0;y<=150;y+=10) {
                     ctx.beginPath();
                     ctx.moveTo(0,y);
                     ctx.lineTo(300,y);
                     ctx.stroke();
              }
              for(var x=0;x<=300;x+=10) {
                     ctx.beginPath();
                     ctx.moveTo(x,0);
                     ctx.lineTo(x,200);
                     ctx.stroke();
              }
       }
              </script>
       </head>
       <body onload="draw();">
              <canvas id="canvas" width="300" height="150">
                     Tarayıcı canvas elemanını desteklemiyor.
```

```
                </canvas>
        </body>
</html>
```

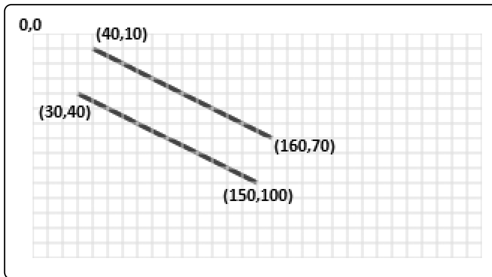
Ekran görüntüsüne görsel olarak bakalım.



`Draw` fonksiyonu içerisindeki kodları aşağıdaki şekilde değiştirip sonuca tekrar bakalım.

```
var draw= function() {
                var canvasEl=document.querySelector("#canvas");
                var ctx=canvasEl.getContext("2d");
                ctx.strokeStyle="#1548ED";
                ctx.lineWidth=3;
                ctx.beginPath();
                ctx.moveTo(30,40);
                ctx.lineTo(150,100);
                ctx.moveTo(40,10);
                ctx.lineTo(160,70);
                ctx.stroke();
                // Izgarayı Oluşturmak için Kodlar…
        }
```

Ekran görüntüsü:



LINETO()

Bu metot varsayılan çizim noktası (başlangıç çizim noktası) ile kendisi ile tanımlanan bitiş noktası arasında düz bir çizgi çizer. `lineTo()` metoduyla bitiş noktasının koordinatları tanımlanır. Başlangıç çizim noktası `moveTo()` metoduyla tanımlanan bir nokta ya da çizim yolunda işaretçinin (*pointer*) bulunduğu noktadır.

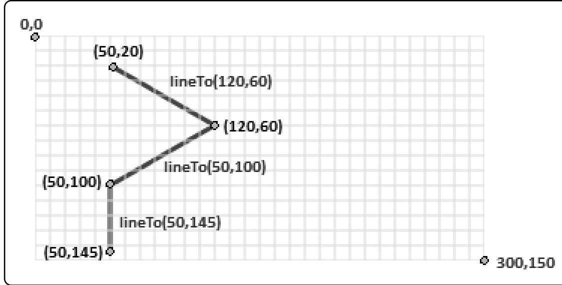
Kullanımı: `context2d.lineTo(x,y)`

`x`, `y` değerleri koordinat sisteminde çizginin bitiş noktasını tanımlar. Bu metot ile tanımlanan çizginin çizdirilmesi için `stroke()` metodu kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>lineTo()</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.strokeStyle="#1548ED";
              ctx.lineWidth=3;
              ctx.beginPath();
              ctx.moveTo(50,20);
              // Çizim için başlangıç noktası tanımladık
              ctx.lineTo(120,60);
              ctx.lineTo(50,100);
              ctx.lineTo(50,145);
              // Tanımladığımız çizgileri çizdirelim.
              ctx.stroke();
              // Izgarayı Oluşturmak için Kodlar…
       }
              </script>
       </head>
       <body onload="draw();">
              <canvas id="canvas" width="300" height="150">
                     Tarayıcı canvas elemanını desteklemiyor.
              </canvas>
       </body>
</html>
```

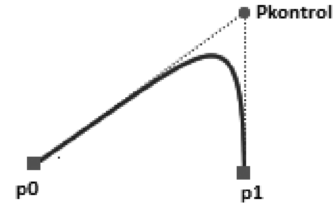
Ekran görüntüsü:



Yukarıdaki resimde çizim noktaları ve hangi metodunun hangi çizgiyi çizdirdiği görülmektedir.

quadraticCurveTo()

Bir kontrol noktası (denetim noktası) kullanarak iki nokta arasında bir eğri çizmeye yarar. İki nokta arasındaki çizgi kontrol noktasına göre eğilerek bir çizim elde edilir.



P_0 başlangıç çizim noktası, P_1 bitiş çizim noktalarıdır. `quadraticCurveTo()` metodu ile kontrol noktası ($P_{kontrol}$) ve bitiş noktası (P_1) tanımlanır.

Başlangıç çizim noktası (P_0) `moveTo()` metoduyla tanımlanan bir nokta ya da çizim yolunda işaretçinin (*pointer*) bulunduğu noktadır.

Kullanımı: `context2d.quadraticCurveTo(cpx, cpy, x, y)`

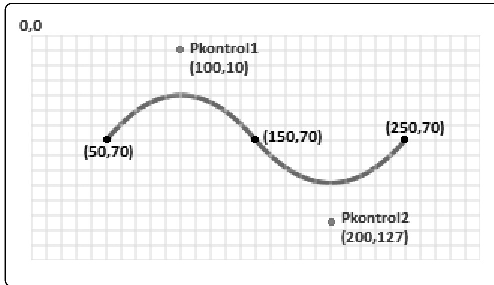
`cpx`, `cpy` değerleri koordinat sisteminde `Pkontrol` noktasını tanımlayan `x` ve `y` değerleridir. Diğer `x`, `y` değerleri ile bitiş noktası (P_1) tanımlanır. Bu metot ile tanımlanan eğrinin çizdirilmesi için `stroke()` metodu kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>quadraticCurveTo()</title>
              <script type="text/javascript">
       var draw= function() {
```

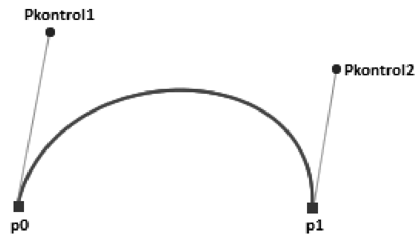
```
                var canvasEl=document.querySelector("#canvas");
                var ctx=canvasEl.getContext("2d");
                ctx.strokeStyle="#F70A0A";
                ctx.lineWidth=3;
                ctx.beginPath();
                ctx.moveTo(50,70);
           ctx.quadraticCurveTo(100,10,150,70);
           ctx.quadraticCurveTo(200,127,250,70);
                ctx.stroke();
                // Izgarayı Oluşturmak için Kodlar…
          }
                </script>
          </head>
          <body onload="draw();">
                <canvas id="canvas" width="300" height="150">
                      Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
          </body>
</html>
```

Ekran görüntüsüne bakalım...



BEZIERCURVETO()

İki kontrol noktası (denetim noktası) ile iki nokta arasında bir eğri çizmeye yarar. Kontrol noktaları ile eğrinin şekli belirlenir.



P_0 başlangıç çizim noktası P_1 bitiş çizim noktasıdır. Kontrol noktaları (P_{k1}, P_{k2}) kullanılarak P_0 ve P_1 noktaları arasında bir eğri oluşturulur. Başlangıç çizim noktası (P_0) `moveTo()`

metoduyla tanımlanan bir nokta ya da çizim yolunda işaretçinin (*pointer*) bulunduğu noktadır. Kontrol noktaları ve bitiş çizim noktası bu metot içerisinde tanımlanır.

Kullanımı: `context2d.bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y)`

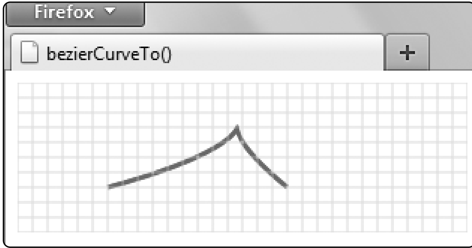
`cp1x`, `cp1y` değerleri `Pkontrol1; cp2x, cp2y` değerleri `Pkontrol2`'yi tanımlar. Diğer `x`, `y` değerleri ile bitiş noktası (P_1) tanımlanır. Bu metot ile tanımlanan eğrinin çizdirilmesi için `stroke()` metodu kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>bezierCurveTo()</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.strokeStyle="#F70A0A";
              ctx.lineWidth=3;
              ctx.beginPath();
              ctx.moveTo(60,70);
              ctx.bezierCurveTo(210,30,100,5,180,70);
              ctx.stroke();
              /*Aşağıdaki kodlar canvas uzerindeki ızgarayı oluşturmak için kullanıldı*/
              ctx.strokeStyle="#A9F5BB";
              ctx.lineWidth=1;
              for(var y=0;y<=150;y+=10) {
                     ctx.beginPath();
                     ctx.moveTo(0,y);
                     ctx.lineTo(300,y);
                     ctx.stroke();
              }
              for(var x=0;x<=300;x+=10) {
                     ctx.beginPath();
                     ctx.moveTo(x,0);
                     ctx.lineTo(x,200);
                     ctx.stroke();
              }
       }
              </script>
```

```
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="300" height="100">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

closePath()

İşaretçinin (*pointer*) bulunduğu nokta ile (İşaretçinin bulunduğu nokta: `beginPath()` metoduyla başlayan serbest çizim alanında çizim işleminde kullanılan en son nokta) `moveTo()` metoduyla tanımlanan çizim işlemine başlangıç noktası arasında düz bir çizgi çizerek çizim yolunu kapatır. Bu yöntemle çizim yolu kapalı bir alana dönüşür. Bu metot tanımlanan çizgileri çizmek için kullanılan `stroke()` metodundan önce kullanılmalıdır.

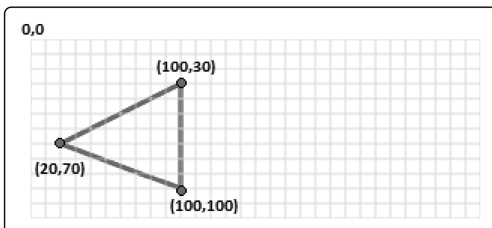
Kullanımı: `context2d.closePath()`

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset="utf-8">
                <title>closePath()</title>
                <script type="text/javascript">
        var draw= function() {
                var canvasEl=document.querySelector("#canvas");
                var ctx=canvasEl.getContext("2d");
                ctx.strokeStyle="#F70A0A";
                ctx.lineWidth=3;
                ctx.beginPath();
```

```
            ctx.moveTo(100,30);
            ctx.lineTo(20,70);
            ctx.lineTo(100,100);
            ctx.closePath();
            ctx.stroke();
            /*Aşağıdaki kodlar canvas üzerindeki ızgarayı oluşturmak için kullanıldı*/
            ctx.strokeStyle="#A9F5BB";
            ctx.lineWidth=1;
            for(var y=0;y<=150;y+=10) {
                    ctx.beginPath();
                    ctx.moveTo(0,y);
                    ctx.lineTo(300,y);
                    ctx.stroke();
            }
            for(var x=0;x<=300;x+=10) {
                    ctx.beginPath();
                    ctx.moveTo(x,0);
                    ctx.lineTo(x,200);
                    ctx.stroke();
            }
        }
            </script>
        </head>
        <body onload="draw();">
            <canvas id="canvas" width="300" height="120">
                    Tarayıcı canvas elemanını desteklemiyor.
            </canvas>
        </body>
</html>
```

Ekran görüntüsü:



ARC ()

Canvas üzerine tanımlanan yarıçapta daire ya da daire dilimi (parçası) çizdirmek için kullanılır. Daire ya da daire parçası yol çizimi ile elde edilir.

```
context2d.arc(x,y,radius,startAngle,endAngle,anticlockwise)
```

`x`, `y` değerleri dairenin merkez noktasının koordinatlarını tanımlamak için kullanılır. `radius` parametresi ile çizdirilecek dairenin yarıçapı ayarlanır. `startAngle` parametresi ile çizdirilecek daire için bir başlangıç açısı, `endAngle` parametresi ile çizdirilecek daire için bir bitiş açısı tanımlanır. Bu değerler aslında yay üzerindeki çizim noktaları olarak da adlandırılabilir. `startAngle` çizime başlangıç noktası, `endAngle` ise çizim bitiş noktasıdır. Bu açı değerleri radyan cinsinden olmalıdır. Derece radyan dönüşümü aşağıdaki formülle yapılabilir.

```
radyan=derece* Math.PI/180
```

`anticlockwise` parametresi ile dairenin saat yönünde mi yoksa tersi yönde mi çizdirileceği ayarlanır. `true` değeri atanırsa ters saat yönünde, `false` değeri atanırsa saat yönünde daire çizimi yapılır. Bu metot ile tanımlanan dairenin kenarlığının çizdirilmesi için `stroke()`, dolgu alanının çizdirilmesi için `fill()` metodu kullanılır.

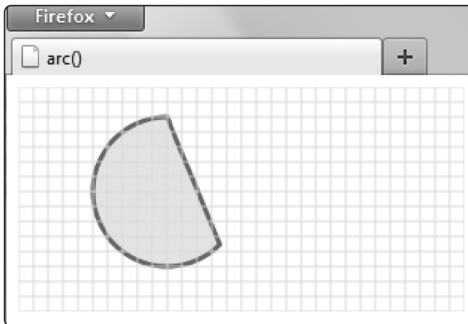
Aşağıdaki örnekle konunun daha iyi anlaşılmasını sağlayalım.

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>arc()</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.strokeStyle="#d64747";
              ctx.fillStyle="#e9e9e9";
              ctx.lineWidth=3;
              ctx.beginPath();
              ctx.arc(100,70,50,1/4*Math.PI,1.5*Math.PI,false);
              ctx.closePath();
              ctx.fill();
              ctx.stroke();
              /*Aşağıdaki kodlar canvas üzerindeki ızgarayı oluşturmak için kullanıldı*/
              ctx.strokeStyle="#A9F5BB";
              ctx.lineWidth=1;
```

```
            for(var y=0;y<=150;y+=10) {
                   ctx.beginPath();
                   ctx.moveTo(0,y);
                   ctx.lineTo(300,y);
                   ctx.stroke();
            }
            for(var x=0;x<=300;x+=10) {
                   ctx.beginPath();
                   ctx.moveTo(x,0);
                   ctx.lineTo(x,200);
                   ctx.stroke();
            }
      }
            </script>
      </head>
      <body onload=”draw();”>
            <canvas id=”canvas” width=”300” height=”150”>
                   Tarayıcı canvas elemanını desteklemiyor.
            </canvas>
      </body>
</html>
```

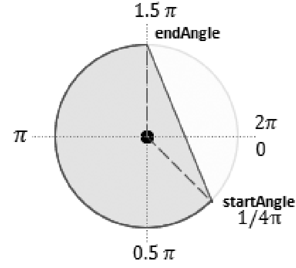
`arc` metodumuza bakarsak merkezi (`100,70`) noktası olan ve `50 px` yarıçapa sahip bir dairenin `45`^{0} (`1/4`π)'den başlayarak `270`^{0} (`1.5`π) dereceye kadar olan bir parçası tanımlanmıştır. `arc()` metodundan sonra kullanılan `closePath()` metodu ile başlangıç ve bitiş noktalarının düz bir çizgi ile birleştirilmesi sağlanmıştır. `fill()` metoduyla tanımlanan daire parçasının dolgusu, `stroke()` metoduyla kenarlığı çizdirilmiştir.

Ekran görüntüsü:



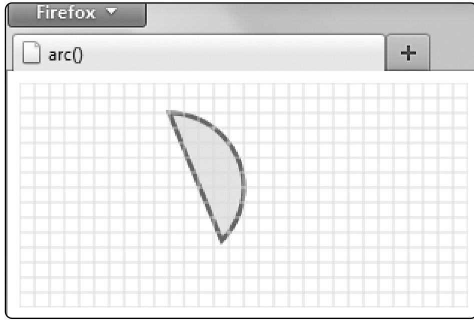
Firefox 4 ekran görüntüsü

Şeklin üzerinde değerleri işaretleyerek anlaşılmasını kolaylaştıralım.



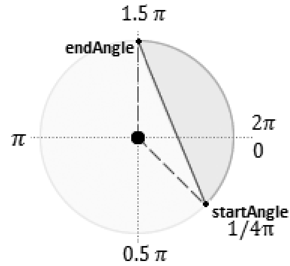
`arc()` metodunda `anticlockwise` parametresi olarak `false` değeri atanmıştır. Bu durumda başlangıç aşısından başlanarak saat yönünde gidilerek bitiş açısına ulaşılır. Bu parametreye atanan değeri `true` olarak değiştirip sonuca tekrar bakalım:

```
ctx.arc(100,70,50,1/4*Math.PI,1.5*Math.PI,true);
```



Firefox 4 ekran görüntüsü

Şeklin üzerinde değerleri işaretleyerek anlaşılmasını kolaylaştıralım.



Bu durumda ise başlangıç aşısından başlanarak ters saat yönünde gidilerek bitiş açısına ulaşılır.

Örnek:

```
<!DOCTYPE html>
<html>
        <head><meta charset="utf-8">
                <title>arc()</title>
                <style type="text/css">
                canvas {
                        border:1px solid gray;
                        /*canvas elemanına bir kenarlık ekledik*/
                }
                </style>
                <script type="text/javascript">
```

```
var draw= function() {
        var canvasEl=document.querySelector("#canvas");
        var ctx=canvasEl.getContext("2d");
        /*canvas elemanı içinde yapılacak çizimler için kenarlık ve dolgu renklerini
        tanımlayalım*/
        ctx.strokeStyle="#d64747";
        ctx.fillStyle="#e9e9e9";
        var coorX=[50,140,230,320];
        /*canvas üzerine 4 tane daire(daire parçası) çizdirilecek bu durumda
        bu dairelerin merkez noktalarının x koordinatlarını bir dizi içerisine atadık*/
        for(var i=0;i<2;i++) {
                var cor=0;
                for(var t=0.5;t<=2;t+=0.5) {
                        ctx.beginPath();
                        /*beginPath() metoduyla yeni bir çizime
                        başlayacağımızı bildiriyoruz.*/
ctx.arc(coorX[cor],(i+0.5)*100,40,0,t*Math.PI,false);
                        /*i=0 ve  t=0.5|1|1.5|2  değerleri için;
                        ctx.arc(50,50,40,0,0.5*Math.PI,false)
                        ctx.arc(140,50,40,0,1*Math.PI,false)
                        ctx.arc(230,50,40,0,1.5*Math.PI,false)
                        ctx.arc(320,50,40,0,2*Math.PI,false)
                        arc() metodunu yukarıdaki şekilde 4 defa
                        çalıştırmış olacağım*/

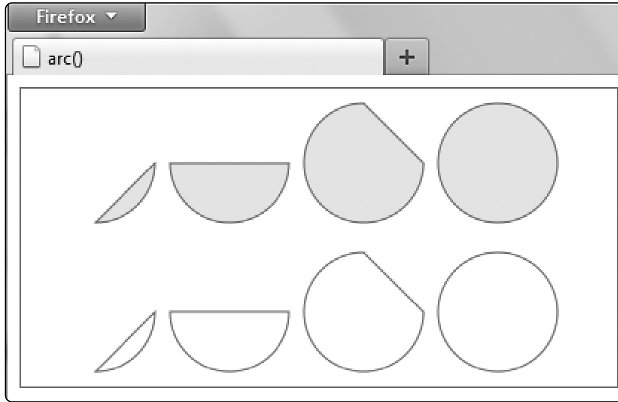
                        /*i=1 ve  t=0.5|1|1.5|2  değerleri için;
                        ctx.arc(50,150,40,0,0.5*Math.PI,false)
                        ctx.arc(140,150,40,0,1*Math.PI,false)
                        ctx.arc(230,150,40,0,1.5*Math.PI,false)
                        ctx.arc(320,150,40,0,2*Math.PI,false)
                        arc() metodunu yukarıdaki şekilde 4 defa
                        çalıştırmış olacağım*/

                        ctx.closePath();
                        /*daire parçası çizimlerinde startAngle ve endAngle
                          noktalarını düz bir çizgi ile birleştirmek için closePah()
                          metodunu kullandım*/

                        if(i==0) {
                                /*Bu döngüde i=0 için arc() metoduyla
                                  tanımlanan dairenin dolgu ve kenarlığı,
                                  i=1 için sadece kenarlığı çizdirilecek*/
```

```
                                        ctx.fill();
                                        ctx.stroke();
                                } else {
                                        ctx.stroke();
                                }
                                cor++;
                        }
                }
        }
                </script>
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="400" height="200">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

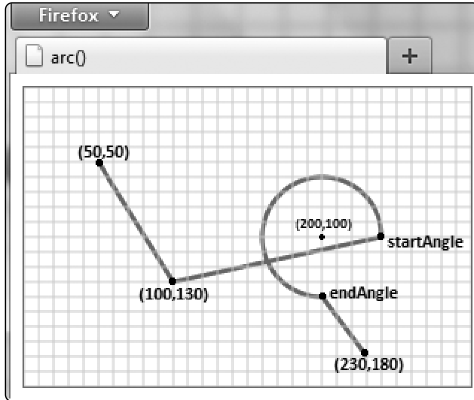
`arc()` metodunu ilk anlatırken *"daire ya da daire parçası, yol çizimi ile elde edilir"* şeklinde bir tanım yapmıştık. Bu şu anlama gelmektedir. Bu metot, eğer çizim yolunda varsayılan bir başlangıç noktası, yani referans noktası varsa; bu başlangıç yani referas noktasından kendisi ile tanımlanan çizime başlangıç noktası (`startAngle`) arasına düz bir çizgi ekleyerek şekli (daire ya da daire parçasını) çizim yoluna ekler. Yukarıdaki örneklerde `arc()` metodunu kullanmadan önce `beginPath()` metoduyla

en son kullanılan, yani işaretçinin bulunduğu nokta bilgisi silinmiştir. Bundan dolayı her çizdirilen daire bir yola bağlanmadan tek başına canvas üzerine çizdirilmiştir. Aşağıdaki örnekle bu açıklamayı daha iyi anlayacaksınız.

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>arc()</title>
              <style type="text/css">
              canvas {
                     border:1px solid gray;
                     /*canvas elemanına bir kenarlık ekledik*/
              }
              </style>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.strokeStyle="#d64747";
              ctx.fillStyle="#e9e9e9";
              ctx.lineWidth=3;
              ctx.beginPath();
              ctx.moveTo(50,50);
              ctx.lineTo(100,130);
              ctx.arc(200,100,40,0,0.5*Math.PI,true);
              ctx.lineTo(230,180);
              ctx.stroke();
              /*Aşağıdaki kodlar canvas üzerindeki ızgarayı oluşturmak için kullanıldı*/
              ctx.strokeStyle="#D8CFFF";
              ctx.lineWidth=1;
              for(var y=0;y<=200;y+=10) {
                     ctx.beginPath();
                     ctx.moveTo(0,y);
                     ctx.lineTo(400,y);
                     ctx.stroke();
              }
              for(var x=0;x<=400;x+=10) {
                     ctx.beginPath();
                     ctx.moveTo(x,0);
```

```
                    ctx.lineTo(x,200);
                    ctx.stroke();
            }
    }
            </script>
    </head>
    <body onload="draw();">
            <canvas id="canvas" width="300" height="200">
                    Tarayıcı canvas elemanını desteklemiyor.
            </canvas>
    </body>
</html>
```

Ekran görüntüsüne şekil üzerinde işaretlemeler yaparak bakalım.



Firefox 4 ekran görüntüsü

> **NOT** `arc()` metodunun dışında iki nokta arasında yarıçapı tanımlanmış bir çember yayı çizdirmek için `arcTo()` metodu kullanılır.

RECT ()

Dikdörtgen için kapalı bir yol oluşturur. Bu metot ile yol çiziminin parçası olan bir dikdörtgen çizdirmiş olursunuz. Bu metot ile oluşturulan dikdörtgen, kendinden önce işaretçinin bulunduğu nokta ile bağlanmaz (Bu metot işaretçiyi `x` ve `y` parametreleri ile belirtilen koordinatlara taşır). Fakat dikdörtgenin sol üst köşesi `rect()` metodundan sonra kullanılan çizimler için başlangıç noktasını temsil eder. Bu metot ile tanımlanan dikdörtgenin `fill()` ve `stroke()` metotları ile kenarlığı ya da dolgusu çizdirilir.

Kullanımı: `context2d.rect(x,y,w,h)`

`x`, `y` parametreleri çizilecek dikdörtgenin sol üst köşesinin koordinatlarını tanımlamak için kullanılır.

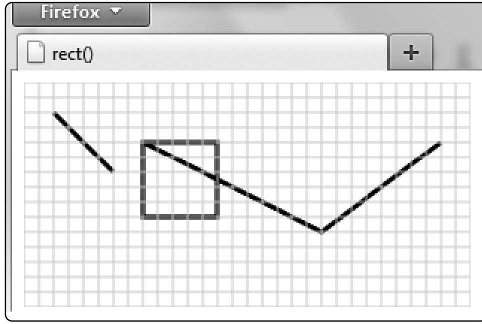
`w`, `h` parametreleri çizilecek dikdörtgenin genişliğini ve yüksekliğini tanımlar.

Örnek:

```
<!DOCTYPE html>
<html>
	<head><meta charset="utf-8">
		<title>rect()</title>
		<script type="text/javascript">
	var draw= function() {
		var canvasEl=document.querySelector("#canvas");
		var ctx=canvasEl.getContext("2d");
		ctx.lineWidth=3;
		ctx.beginPath();
		ctx.moveTo(20,20);
		ctx.lineTo(60,60);
		ctx.rect(80,40,50,50);
		ctx.lineTo(200,100);
		ctx.lineTo(280,40);
		ctx.stroke();
		/*Izgara*/
		ctx.strokeStyle="#D8CFFF";
		ctx.lineWidth=1;
		for(var y=0;y<=150;y+=10) {
			ctx.beginPath();
			ctx.moveTo(0,y);
			ctx.lineTo(300,y);
			ctx.stroke();
		}
		for(var x=0;x<=300;x+=10) {
			ctx.beginPath();
			ctx.moveTo(x,0);
			ctx.lineTo(x,150);
			ctx.stroke();
		}
	}
```

```
                </script>
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="300" height="150">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

FILL() VE STROKE()

Yol çizimi metotları ile tanımlanan çizgilerin çizdirilmesi için `stroke()` yol çizimi ile oluşturulan kapalı alanların dolgusunun çizdirilmesi için `fill()` metodu kullanılır. Kısacası; serbest yol oluşturmak için kullanılan metotlar sadece tanımlama yapar. Bu metotlarla tanımlanan çizgi ya da şekillerin canvas üzerine çizilmesi için `stroke()` ve `fill()` metotları kullanılır. Bu metotlar kendilerinden önce tanımlanan `strokeStyle`, `fillStyle` özelliklerine atanan renk değerlerini ve diğer çizgi stil özelliklerini kullanarak (eğer özellikler tanımlanmamış ise varsayılan değerleri kullanır) belirtilen çizgiyi ya da dolguyu canvas üzerinde oluştururlar.

Kullanımları:

```
context2d.fill()
context2d.stroke()
```

Bu metotlar kendilerinden önce `beginPath()` metoduyla başlanıp, oluşturulmuş serbest yol çizimi için geçerlidirler.

Aşağıdaki örneği inceleyiniz.

```
ctx.beginPath(); /*1.çizim*/
ctx.moveTo(20,20);
ctx.lineTo(60,60);

ctx.beginPath(); /*2.çizim*/
ctx.moveTo(80,80);
ctx.lineTo(200,100);
ctx.lineTo(280,40);
ctx.stroke();
ctx.fill();
```

Bu durumda `stroke()`, `fill()` metotları sadece ikinci `beginPath()` metoduyla başlayan yol tanımlaması için geçerli olurlar. Yol çizimi ile ilgili bir özelliğin kullanılabilmesi ya da bir metodun çizim tanımlaması yapması için `stroke()` ve `fill()` metotlarından önce tanımlanmış olması gerekir.

CLIP()

Canvas üzerinde çizim yapılmış alanın dışında kalan kısmı kırparak sadece içeriği barındıran bir alan elde edilir. Bu metot kullanıldıktan sonra canvas üzerinde sadece kırpma sonucu elde edilmiş bu alan gösterilir (Daha sonra çizdirilen grafiklerin sadece bu alan içindeki kısmı gösterilir).

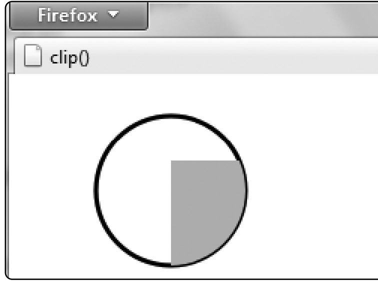
Kullanımı: `context2d.clip()`

Örnek:

```
<!DOCTYPE html>
<html>
      <head><meta charset="utf-8">
            <title>clip()</title>
            <script type="text/javascript">
      var draw= function() {
            var canvasEl=document.querySelector("#canvas");
            var ctx=canvasEl.getContext("2d");
            ctx.lineWidth=3;
            ctx.fillStyle="#FFA666";
            ctx.beginPath();
            ctx.arc(100,70,50,0,2*Math.PI,true);
            ctx.stroke();
            ctx.clip();
```

```
/*canvas üzerinde sadece çember var ve clip() metodu kullanıldı. Bu durumda çember dışında
kalan alanlar kırpılır. Bu metodtan sonra sadece çember ve içerisine çizdirilecek grafikler gösterilir.*/
                ctx.fillRect(100,50,60,70);
        }
                </script>
        </head>
        <body onload="draw();">
                <canvas id="canvas" width="300" height="150">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Dikkat ederseniz `clip()` metodundan sonra çizdirilen dikdörtgenin sadece çember içinde kalan kısmı gösterildi.

Gradient ve Pattern Metotları

Aşağıdaki metotları kullanarak renk geçisi içeren ya da bir resim ya da videonun tekrarından oluşan (resim ya da video dolgu malzemesi olarak kullanılır) dolgular oluşturabilirsiniz ve oluşturduğunuz bu dolguları canvas üzerinde çizdirdiğiniz şekillere dolgu ya da kenarlık rengi olarak atayabilirsiniz.

addColorStop()

`createLinearGradient()` ya da `createRadialGradient()` metotları tanımlandığında oluşan renk geçişi nesnesi bir `gradient-line`'a sahiptir. `gradient-line` renk geçişinin hangi renkle ve hangi noktada olacağını tanımlamak için kullanılan bir referans çizgisidir.

Kullanımı: `gradientObject.addColorStop(offset,color)`

`offset` parametresi, tanımlanan rengin `gradient-line` üzerinde hangi noktada tam olarak gösterileceğini ayarlar. Bu şekilde birden fazla renk `gradient-line` üzerinde tanımlanarak renk geçişi elde edilmiş olur. `offset` özelliğine `0.0` ile `1.0` arasında bir değer atanabilir.

`color` parametresi, `gradinet-line` üzerinde `offset` parametresi ile tanımlanan noktada gösterilecek rengi ayarlar. CSS color tipinde bir renk değeri olabilir.

createLinearGradient()

Bu metod doğrusal renk geçişini temsil eden bir nesne oluşturur.

Kullanımı: `gradient=context2d.createLinearGradient(x0,y0,x1,y1)`

Bu metodunun geriye döndürdüğü `gradient` değişkeni `CanvasGradient` tipinde bir nesnedir. (`x0,y0`) parametreleri ile `gradient-line` (*referans çizgisinin*) başlangıç noktasının, (`x1,y1`) parametreleri ile referans çizgisinin bitiş noktasının koordinatları ayarlanır.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>createLinearGradient()</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.lineWidth=3;
              *linearGradient Nesnesi oluşturalım ve gradient-line üzerinde geçişte
              kullanılacak renkleri ve konumlarını ayarlayalım*/
              var lGradient=ctx.createLinearGradient(30,30,270,170);
              lGradient.addColorStop("0","crimson");
              lGradient.addColorStop("1.0","white");
              ctx.fillStyle=lGradient;
              ctx.fillRect(50,50,200,100);
       }
              </script>
       </head>
```

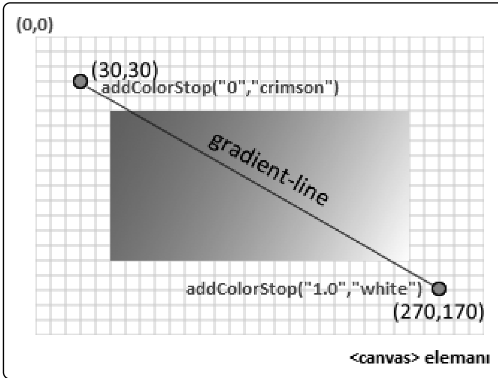
```
	<body onload="draw();">
		<canvas id="canvas" width="300" height="200">
			Tarayıcı canvas elemanını desteklemiyor.
		</canvas>
	</body>
</html>
```

Ekran görüntüsü:



Bu ekran görüntüsü Firefox 4 ile elde edilmiştir.

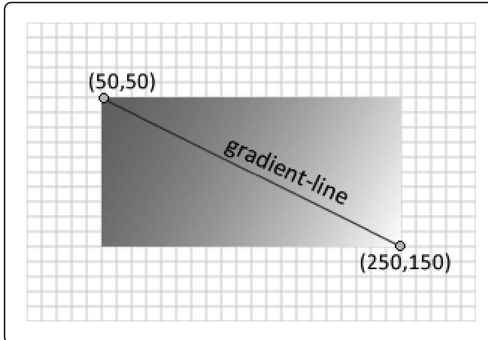
Resim üzerinde işaretlemeler yaparak, `gradient-line` konumunu ve tanımlanan renk değerlerini görsel olarak görelim.



`gradient-line` referans çizgisi `canvas(0,0)` noktasına göre doğrusal renk geçişi nesnesi (`createLinearGradient`) oluşturulurken tanımlanan koordinatlar baz alınarak oluşturulur. Bu gradient-line, sadece belirtilen dolguyu kullanan çizimler için geçerli olur.

Yukarıdaki örnekte aşağıdaki değişiklikleri yapıp sonuca görsel olarak tekrar bakalım.

```
var lGradient=ctx.createLinearGradient(50,50,250,150);[Değiştirin]
```



Örnek:

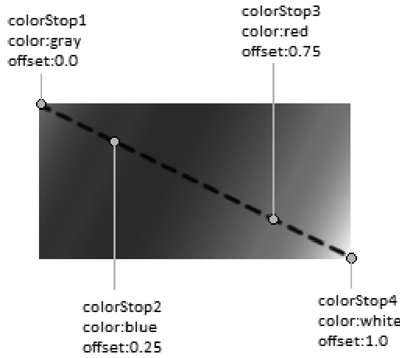
```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>createLinearGradient()</title>
              <script type="text/javascript">
       var draw= function() {
              var canvasEl=document.querySelector("#canvas");
              var ctx=canvasEl.getContext("2d");
              ctx.lineWidth=3;
              var lGradient=ctx.createLinearGradient(50,50,250,150);
              lGradient.addColorStop("0.0","gray");
              lGradient.addColorStop("0.25","blue");
              lGradient.addColorStop("0.75","red");
              lGradient.addColorStop("1","white");
              ctx.fillStyle=lGradient;
              ctx.fillRect(50,50,200,100);
       }
              </script>
       </head>
       <body onload="draw();">
              <canvas id="canvas" width="300" height="200">
                     Tarayıcı canvas elemanını desteklemiyor.
              </canvas>
       </body>
</html>
```

Ekran görüntüsü:



Bu ekran görüntüsü Firefox 4 ile elde edilmiştir.

Daha iyi anlamanız için şekil üzerinde işaretlemeler yapalım.



createRadialGradient()

Bu metod dairesel renk geçişini temsil eden bir nesne oluşturur.

Kullanımı: `gradient=context2d.createRadialGradient(x0,y0,r0,x1,y1,r1)`

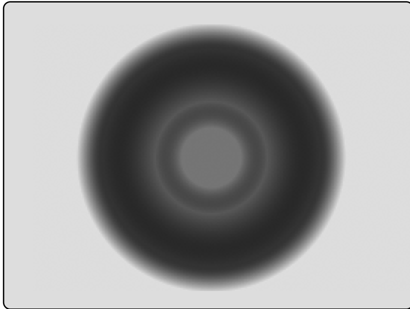
Bu metodunun geriye döndürdüğü gradient değişkeni `CanvasGradient` tipinde bir nesnedir. Renk geçişi merkez çemberin yayından başlayarak dairesel olarak dış çemberin yayına kadar devam eder. (`x0, y0`) parametreleri ile birinci çemberin (*iç çember*) merkez noktası, `r0` parametresi ile birinci çemberin yarıçapı, (`x1, y1`) parametreleri ile ikinci çemberin (*dış çember*) merkez noktası, `r1` parametresi ile ikinci çemberin yarıçapı ayarlanır. Düzgün dairesel renk geçişleri oluşturmak için iç ve dış çemberin merkez noktalarının aynı olması önemlidir.

`gradient-line` (*referans çizgisinin*) başlangıç noktası merkez çemberin yayı bitiş noktası dış çemberin yayıdır.

Örnek:

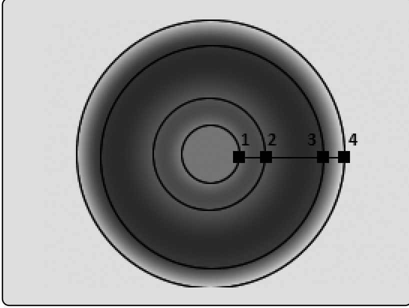
```
<!DOCTYPE html>
<html>
	<head><meta charset="utf-8">
		<title>createRadialGradient()</title>
		<script type="text/javascript">
	var draw= function() {
		var canvasEl=document.querySelector("#canvas");
		var ctx=canvasEl.getContext("2d");
		// radialGradient Nesnesi oluşturalım;
		var rGradient=ctx.createRadialGradient(150,100,20,150,100,90);
		rGradient.addColorStop("0.0","#FA2AB1");
		rGradient.addColorStop("0.25","green");
		rGradient.addColorStop("0.75","blue");
		rGradient.addColorStop("1.0","khaki");
		ctx.fillStyle=rGradient;
		ctx.fillRect(10,10,280,180);
	}
		</script>
	</head>
	<body onload="draw();">
		<canvas id="canvas" width="300" height="200">
			Tarayıcı canvas elemanını desteklemiyor.
		</canvas>
	</body>
</html>
```

Ekran görüntüsü:



Bu ekran görüntüsü Firefox 4 ile elde edilmiştir.

Daha iyi görebilmeniz için şekil üzerinde işaretlemeler yapalım.



Renk geçişi için kullanılacak `gradient-line` referans çizgisi yukarıdaki şekilde görülmektedir.

1. colorStop1 color: #FA2AB1 offset:**0.0**	**2.** colorStop2 color: green offset:**0.25**	**3.** colorStop3 color: blue offset:**0.75**	**4.** colorStop4 color:khaki offset:**1.0**

Aşağıdaki örnekte form elemanlarını kullanarak tanımlanan dairesel renk geçişi dolgusu, canvas elemanı içerisinde Mouse işaretçisini takip edecek. Sayfa ilk yüklendiğinde canvas merkezinde varsayılan değerlerle oluşturulacak bu dairesel renk geçişi alanı kullanıcının form elemanlarındaki değerleri değiştirmesine bağlı olarak değişebilecek ve Mouse işaretçisini takip edecek.

```
<!DOCTYPE html>
<html>
	<head><meta charset=”utf-8”>
		<title>createRadialGradient()</title>
		<style type=”text/css”>
	body {
		margin:0px;
	}
	div#edit {
		position:absolute;
		left:0px;
		top:400px;
		background-color:lightblue;
		width:500px;
		font-family:verdana;
		font-size:12px;
```

```
        height:50px;
}
div#edit input[type='number'] {
        width:40px;
}
        </style>
        <script type="text/javascript">
var draw= function() {
        var canvasEl=document.querySelector("#canvas");
        var ctx=canvasEl.getContext("2d");
        /*canvas elemanına ulaştık ve bu elemanı 2 boyutlu bir çizim
        alanına dönüştürdük*/
        ctx.fillStyle=createGradient(250,200,ctx);
        /*createGradient() fonksiyonu geriye (250,200) noktası merkezli
        radial gradient nesnesi döndürür.*/
        ctx.fillRect(0,0,500,400);
        /*Sayfa yüklendiğinde canvas elemanı üzerinde dairesel renk geçişi
        oluşturmak için canvas boyutlarında bir dikdörtgen çizdirdik. Sadece dolgu
        alanına sahip olan bu dikdörtgen fillStyle özelliğine atanan dairesel renk
        geçişi dolgusunu kullanacak.*/
}
var createRadialgr= function(event) {
        /*canvas üzerinde kullanıcı fareyi hareket ettirdiğinde çalışacak fonksiyon*/
        var canvasEl=document.querySelector("#canvas");
        var ctx=canvasEl.getContext("2d");
        var evt=event||window.event;
        /*canvas içerisinde dairesel renk geçişi fareyi takip edeceğinden mouse'un
        canvas içerisindeki pozisyonunu bilmemiz gerekir. Internet Explorer olay
        nesnesini window.event şeklinde tanımlarken diğer tarayıcılar sadece event
        şeklinde tanımlar. Bu durumda eğer tarayıcı event nesnesini destekliyorsa
        evt=event eğer desteklemiyorsa evt=window.event olacaktır.
        var evt=event|| window.event ifadesinde mantıksal(logical Or) kullanılmıştır.
        Logical Or
        result=operant1 || operant2
        *operant1 object ise result=operant1
        *operant1 false ise result=operant2
        */

        var xkor=evt.clientX;
        var ykor=evt.clientY;
        /*clientX mouse işaretçisi ile tarayıcı sol köşesi arasındaki uzaklığı,
```

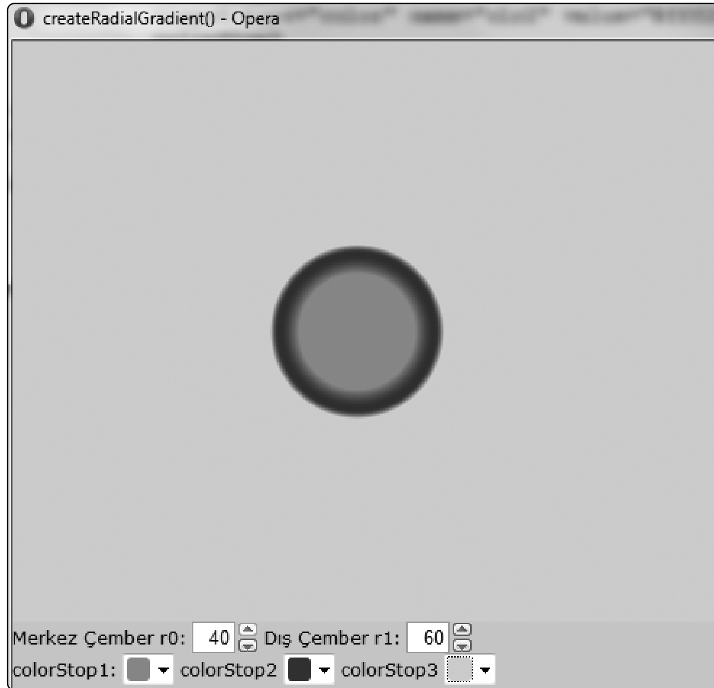
```
                clientY mouse işaretçisi ile tarayıcı üst köşesi arasındaki uzaklığı verir.
                Bu durumda bizim yapmak istediğimiz mouse işaretçisinin canvas içerisindeki
                konumunu merkez olarak kabul eden dairesel bir gradient oluşturmaktır.
                body{margin:0}tanımlaması yaptığımızdan xkor, ykor değişkenleri
                mouse'un canvas içerisindeki konumunu vermektedir.*/
                ctx.fillStyle=createGradient(xkor,ykor,ctx);
                /*mouse işaretçisinin bulunduğu noktayı merkez nokta olarak kabul eden
                  dairesel bir gradient oluşturulup fillStyle özelliğine atanıyor*/

                ctx.fillRect(0,0,500,400);
                /*Son olarak yukarıda tanımlanan dairesel renk geçişi dolgusunu
                  kullanacak ve canvasın tamamını kapsayacak bir dikdörtgen çizdirdik*/
        }
        var createGradient= function(xkor,ykor,ctx) {
                /*Bu fonksiyon xkor,ykor değişkenlerine atanan değerleri ve canvas elemanının
                  altında tanımlanan form elemanları içerisindeki verileri kullanarak
                  radialGradient nesnesi oluşturmak için tanımlanmıştır.*/

                var number1=document.querySelector("input[name='num1']");
                var number2=document.querySelector("input[name='num2']");
                var color1=document.querySelector("input[name='clr1']");
                var color2=document.querySelector("input[name='clr2']");
                var color3=document.querySelector("input[name='clr3']");
                var rGradient=ctx.createRadialGradient(xkor,ykor,
number1.value,xkor,ykor,number2.value);
                rGradient.addColorStop("0.0",color1.value);
                rGradient.addColorStop("0.75",color2.value);
                rGradient.addColorStop("1.0",color3.value);
                return rGradient;
        }
                </script>
        </head>
        <body onload="draw();" >
                <canvas id="canvas" width="500" height="400"
onmousemove="createRadialgr(event);">
                        Tarayıcı canvas elemanını desteklemiyor.
                </canvas>
                <div id="edit">
                        Merkez Çember r0:
                        <input type="number" min="0" max="40" step="5"
value="0" name="num1"/>
```

```
                              Dış Çember r1:
                              <input type=”number” min=”10” value=”60”
max=”100” step=”5” name=”num2”/>
                              <br/>
                              colorStop1:
                              <input type=”color” name=”clr1” value=”#ed1c24” />
                              colorStop2
                              <input type=”color” name=”clr2” value=”#fff200” />
                              colorStop3
                              <input type=”color” name=”clr3” value=”#a349a4” />
                  </div>
        </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

createPattern()

Resim ya da video elemanlarının tekrarından oluşan dolgular oluşturmak için kullanılır. Bu meot canvasPattern tipinde bir nesne oluşturur.

Kullanımları:

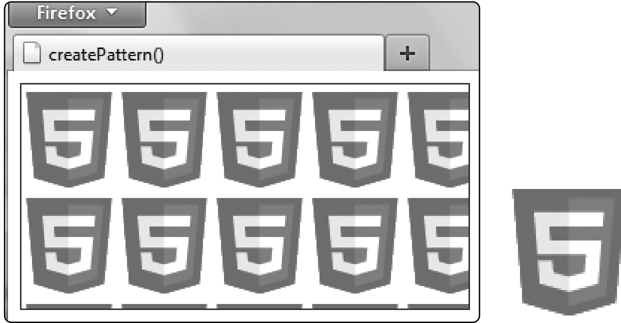
```
pattern=context2d.createPattern(HTMLImageElement,repetition)
pattern=context2d.createPattern(HTMLVideoElement,repetition)
```

repetion parametresi repeat (dolgu malzemesi tekrar eder), repeat-x (dolgu malzemesi yatay eksende tekrar eder), repeat-y (dolgu malzemesi dikey eksende tekrar eder), no-repeat (dolgu malzemesi tekrar etmez) string değerlerinden birini alır.

Örnek:

```
<!DOCTYPE html>
<html>
       <head><meta charset="utf-8">
              <title>createPattern()</title>
              <style type="text/css">
              canvas{
                border:1px solid blue;
              }
              </style>
              <script type="text/javascript">
       var draw= function() {
              var canvas=document.getElementsByTagName("canvas")[0];
              var context2d=canvas.getContext("2d");
              var image=new Image();
              image.src="html5.png";
              var pattern=context2d.createPattern(image,"repeat");
              context2d.fillStyle=pattern;
              context2d.fillRect(0,0,canvas.width,canvas.height);
       }
              </script>
       </head>
       <body onload="draw();" >
              <canvas id="canvas" width="300" height="150" >
                     Tarayıcı canvas elemanını desteklemiyor.
              </canvas>
       </body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü. html5.png
Dolgu malzemesi (resmi)

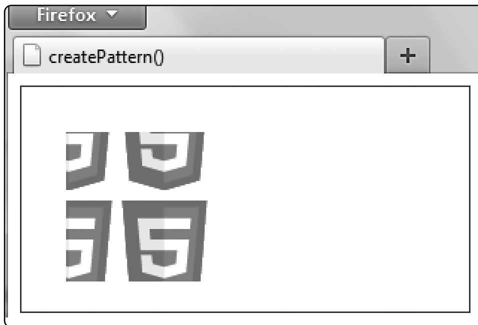
Canvas üzerine çizdirdiğimiz dikdörtgen (canvas elemanının tamamını kaplayan) `fillStyle` özelliği ile tanımlanan ve resmin tekrarından oluşan dolguyu kullanmıştır.

Burada şunu belirtelim *html5.png* resmi kullanılarak oluşturulan dolgu `fillRect()` metodu tarafından kullanılmıştır.

Yukarıdaki örnekte aşağıdaki değişikliği yapıp konumuza devam edelim.

```
context2d.fillRect(30,30,100,100);
```

Ekran görüntüsü:

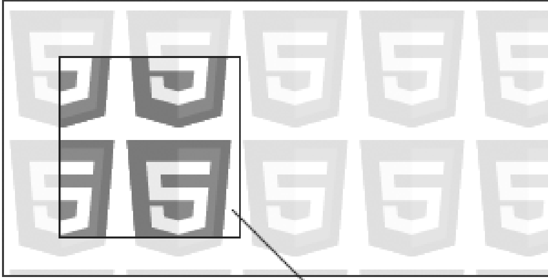


Firefox 4 ekran görüntüsü

Sonuç sizi şaşırtmış olabilir. Çünkü biz resmin tekrarından oluşan dolguyu canvas üzerine çizdirilecek grafikler için tanımladık ve beklentimiz şu idi; resim çizdirdiğimiz dikdörtgenin sol üst köşesinden başlayıp tekrar etmeli. Fakat `createPattern()` metodu `canvas(0,0)` konumdan başlayarak kendisine verilen `repetion` parametre-

sine bağlı olarak dolguyu tanımlar. Yani aslında bu dolgu canvas elemanının tamamına göre tanımlanır. Fakat canvas üzerine çizdirilen grafikler konumlarına göre canvas üzerine tanımlanmış bu dolguyu kullanırlar.

Resme işaretlemeler yaparak yakından bakalım.



fillRect(30,30,100,100) metoduyla çizdirilen dikdörtgen



Bu metodu oluşturduğunuz resmin `onload` olayında kullanırsanız özellikle Firefox tarafında çıkan problemi yaşamazsınız. Yukarıdaki kodu aşağıdaki gibi yazmanız mümkündür.

```
var draw= function() {
            var canvas=document.getElementsByTagName("canvas")[0];
            var context2d=canvas.getContext("2d");
            var image=new Image();
            image.src="html5.png";
      image.onload = function(){
            var pattern=context2d.createPattern(image,"repeat");
            context2d.fillStyle=pattern;
            context2d.fillRect(0,0,canvas.width,canvas.height);
      }
}
```

Transformation Metotları

Canvas üzerine çizdirdiğiniz grafikler için transformasyon metotlarını kullanarak şekilleri tekrar boyutlandırabilir, döndürebilir, koordinatlarını değiştirebilir ya da eğebilirsiniz. Şimdi bu metotları sırasıyla görelim.

SCALE()

Ölçeklendirme (yeniden boyutlandırma) işlemleri için kullanılır. Kendisinden sonra tanımlanan çizim metotlarının kullanacağı yatay ve dikey olmak üzere iki tane ölçeklendirme katsayısı tanımlar.

Kullanımı: `context2d.scale(x,y)`

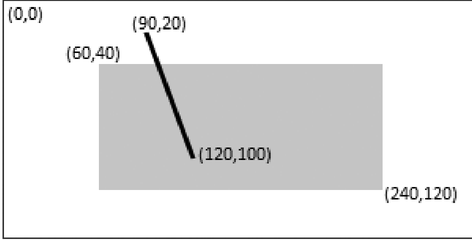
`x` parametresi çizim metotları için tanımlanan yatay değerler için bir ölçekleme katsayı tanımlarken, `y` parametresi dikey değerler için bir ölçekleme katsayısı tanımlar. Bu metot kendisinden sonra tanımlanan çizim metotları için geçerlidir.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>scale()</title>
    <style type="text/css">
        canvas
        {
            border: 1px solid blue;
        }
    </style>
    <script type="text/javascript">
        var draw = function () {
            var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
            ctx.fillStyle = "lightblue";
            ctx.scale(3, 2);
            ctx.fillRect(20, 20, 60, 40);
            ctx.beginPath();
            ctx.moveTo(30, 10);
            ctx.lineTo(40, 50);
            ctx.stroke();
        }
    </script>
</head>
<body onload="draw();">
    <canvas id="canvas" width="300" height="150">
             Tarayıcı canvas elemanını desteklemiyor.
```

```
    </canvas>
</body>
</html>
```

Sonuca görsel işaretlemeler yaparak bakalım.



Şimdi kod satırlarımıza tekrar bakalım.

```
ctx.scale(3, 2);
```

satırı ile yatay eksende (x ekseninde) tanımlanacak değerler için `3` katsayısı, dikey eksende (y ekseninde) tanımlanacak değerler için `2` katsayısı tanımlanmıştır.

```
ctx.fillRect(20, 20, 60, 40);
```

satırı ile canvas üzerine bir dikdörtgen çizdirmek istiyoruz. İlk `20` değeri ve `60` değeri yatay eksen üzerinde tanımlanmış değerlerdir. Bu durumda tarayıcı bu değerleri `3` ile çarpar. İkinci `20` değeri ve `40` değeri dikey eksende tanımlanmış değerler olduğundan tarayıcı bu değerleri `2` ile çarpar ve sonuçta bulunan değerlere göre dikdörtgen çizdirilir.

Yeni değerlerle metodun son hali `ctx.fillRect(60, 40, 180, 80)` şeklinde olacaktır:

```
ctx.moveTo(30, 10)
```

`moveTo()` metodu; yeni bir çizim başlangıç noktası tanımlamak için kullanılır. `context2d.moveTo(x,y);` Bu durumda tarayıcı `30` değerini `3` ile çarpar, `10` değerini `2` ile çarpar ve yeni bulunan değerlere göre başlangıç noktasını tanımlar. Ölçekleme sonucunda metodun son hali `ctx.moveTo(90, 20)` şeklinde olacaktır.

```
ctx.lineTo(40, 50);
```

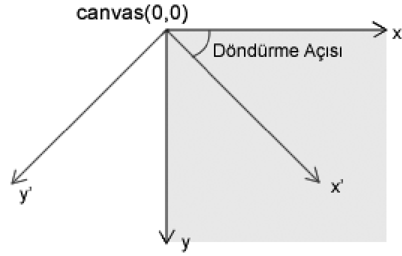
Ölçekleme sonucu tarayıcı tarafından kullanılacak metodun son hali:

```
ctx.stroke();
```

ROTATE()

Döndürme işlemleri için kullanılır. Kendisinden sonra tanımlanan çizim metotlarının kullanacağı bir döndürme açısı tanımlar. Döndürme işlemi saat yönünde yapılır ve döndürme işleminde kullanılan orijin noktası `canvas(0,0)`'dır.

Aslında daha iyi kavrayabilmeniz için şöyle düşünebilirsiniz; `rotate()` metodundan sonra canvas koordinat sistemi belirtilen açı kadar döndürülür ve şekiller, yeni oluşan bu koordinat sistemine göre çizdirilir. Yandaki resimle ne demek istedimi daha iyi anlayacaksınız. `x'` ve `y'` döndürme sonucunda oluşan yeni canvas koordinat sistemi eksenlerini tanımlar.



Buradan canvas elemanı döndürülür şeklinde bir sonuç çıkaramayız. Döndürülen canvas sol üst köşesini orijin olarak kabul eden canvas koordinat sistemidir.

Kullanımı: `context2d.rotate(angle)`

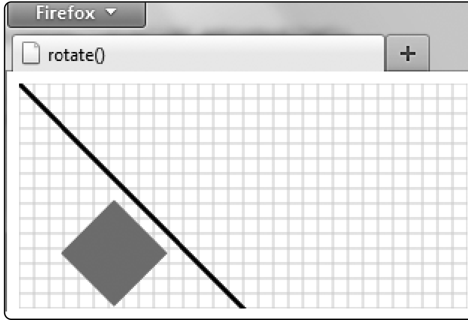
`angle` parametresi radyan cinsinden döndürme açısını temsil eder.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>rotate()</title>
    <script type="text/javascript">
        var draw = function () {
            var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
            ctx.fillStyle = "red";
            /*Aşağıdaki döngüler canvas üzerindeki ızgarayı oluşturmak için kullanılmıştır.
              rotate() metodu tanımlandığı satırdan sonra tanımlanan metotlar için
              geçerlidir.*/
            ctx.strokeStyle = "#D8CFFF";
            ctx.lineWidth = 1;
            for (var y = 0; y <= 150; y += 10) {
                ctx.beginPath();
                ctx.moveTo(0, y);
```

```
                ctx.lineTo(300, y);
                ctx.stroke();
            }
            for (var x = 0; x <= 300; x += 10) {
                ctx.beginPath();
                ctx.moveTo(x, 0);
                ctx.lineTo(x, 150);
                ctx.stroke();
            }
            ctx.rotate(Math.PI / 4); // 450
            /*rotate() metodu aşağıdaki metotların çizdireceği grafikler için geçerlidir*/
            ctx.strokeStyle = "black";
            ctx.lineWidth = 3;
            ctx.beginPath();
            ctx.moveTo(0, 0);
            ctx.lineTo(300, 0);
            ctx.lineTo(300, 150);
            ctx.lineTo(0, 150);
            ctx.closePath();
            ctx.stroke();
            ctx.fillRect(100, 10, 50, 50);
        }
    </script>
</head>
<body onload="draw();">
       <canvas id="canvas" width="300" height="150">
              Tarayıcı canvas elemanını desteklemiyor.
       </canvas>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

TRANSLATE()

Çizim metotlarının orijin noktası olarak canvas elemanın sol üst köşesini (0,0) kullandığını biliyorsunuz. Çizim metotları tarafından varsayılan olarak kullanılan bu orijin noktasını değiştirmek için `translate()` metodu kullanılır. `translate()` metodu ile tanımlanan yeni nokta çizim metotları tarafından canvas orijin noktası (0,0) olarak kullanılır.

Kullanımı: `context.translate(x,y)`

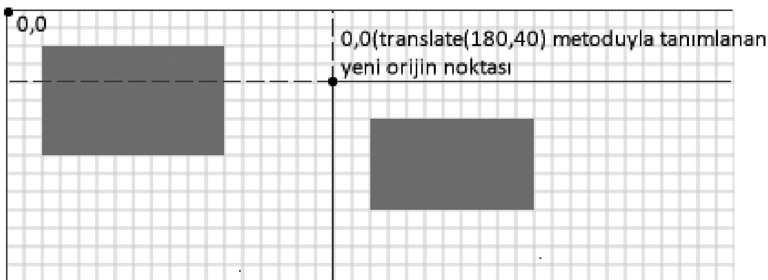
x, y parametreleri canvas(0,0) sol üst köşesine göre (varsayılan orijin noktası) yeni bir orijin noktası tanımlamak için kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>translate()</title>
    <script type="text/javascript">
        var draw = function () {
            var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
            ctx.fillStyle = "red";
            ctx.strokeStyle = "#D8CFFF";
            ctx.lineWidth = 1;
            for (var y = 0; y <= 150; y += 10) {
                ctx.beginPath();
```

```
                    ctx.moveTo(0, y);
                    ctx.lineTo(400, y);
                    ctx.stroke();
               }
               for (var x = 0; x <= 400; x += 10) {
                    ctx.beginPath();
                    ctx.moveTo(x, 0);
                    ctx.lineTo(x, 150);
                    ctx.stroke();
               }
               ctx.fillRect(20, 20, 100, 60);
               // Yeni bir orijin noktası tanımlayalım;
               ctx.translate(180, 40);
               // Yeni orijin noktasına göre bir dikdörtgen çizdirelim...
               ctx.fillRect(20,20,90,50);
          }
     </script>
</head>
<body onload="draw();">
       <canvas id="canvas" width="400" height="150">
              Tarayıcı canvas elemanını desteklemiyor.
       </canvas>
</body>
</html>
```

Ekran görüntüsüne üzerinde işaretlemeler yaparak bakalım.



TRANSFORM(), SETTRANSFORM()

`transform()` metodu, transformasyon matrisi (*affine transform*) ile grafikler üzerinde dönüşüm işlemleri yapmak için kullanılır.

`setTransform()` metodu, `transform()` metodu ile aynı parametreleri alır ve transformasyon matrisini varsayılan haline geri döndürdükten (resetledikten) sonra parametre olarak girilen matris değerleri ile yeni bir dönüşüm işlemi yapar.

Aşağıda, bu metotlar ile kullanılacak (`3*3`) matris görünmektedir.

m11	m21	dx
m12	m22	dy
0	0	1

```
context2d.transform(m11,m12,m21,m22,dx,dy)
context2d.setTransform(m11,m12,m21,m22,dx,dy)
```

Aşağıda temel transform işlemleri için kullanılacak matris formatları verilmiştir.

- **Rotation**

$\cos(\alpha)$	$-\sin(\alpha)$	0
$\sin(\alpha)$	$\cos(\alpha)$	0
0	0	1

```
context2d.transform(cos(a),sin(a),-sin(a),cos(a),0,0)
```

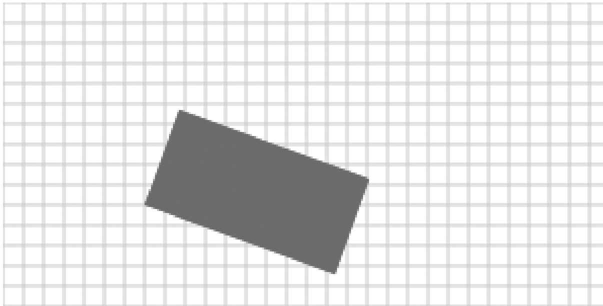
`a` parametresi, döndürme açısıdır.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>transform()</title>
    <script type="text/javascript">
        var draw = function () {
            var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
            ctx.fillStyle = "red";
            ctx.strokeStyle = "#D8CFFF";
            ctx.lineWidth = 1;
            for (var y = 0; y <= 150; y += 10) {
                ctx.beginPath();
                ctx.moveTo(0, y);
```

```
                    ctx.lineTo(300, y);
                    ctx.stroke();
               }
               for (var x = 0; x <= 300; x += 10) {
                    ctx.beginPath();
                    ctx.moveTo(x, 0);
                    ctx.lineTo(x, 150);
                    ctx.stroke();
               }
               var sin = Math.sin(Math.PI / 9);
               var cos = Math.cos(Math.PI / 9);
               ctx.transform(cos, sin, -sin, cos, 0, 0);
               ctx.fillRect(100, 20, 100, 50);
          }
     </script>
</head>
<body onload="draw();">
       <canvas id="canvas" width="300" height="150">
              Tarayıcı canvas elemanını desteklemiyor.
       </canvas>
</body>
</html>
```

Ekran görüntüsü:



Döndürme işleminde orijin noktası varsayılan olarak canvas(0,0) olacaktır.

- **Scale**

sx	0	0
0	sy	0
0	0	1

```
context2d.transform(sx,0,0,sy,0,0)
```

`sx` parametresi, çizim metotları için tanımlanan yatay değerler için bir ölçekleme katsayı tanımlarken, `sy` parametresi dikey değerler için bir ölçekleme katsayısı tanımlar.

- **Translate**

1	0	tx
0	1	ty
0	0	1

```
context2d.transform(1,0,0,1,tx,ty)
```

`tx`, `ty` parametreleri `canvas(0,0)` sol üst köşesine göre (varsayılan orijin noktası) yeni bir orijin noktası tanımlamak için kullanılır. Yukarıdaki transformasyon matris kullanılarak **eğme** (*skew*) işlemleri de yapılabilir.

Canvas Üzerinde Metin İşlemleri

`Context2d` nesnesi üzerinde metin alanları oluşturabiliriz. `context2d` nesnesinin tanımladığı metin özellikleri ve metotlarına bakalım.

font, textAlign ve textBaseline

Aşağıdaki özellikler tanımlandıkları satırdan sonra geçerli olacaklardır. Aşağıdaki özellikler okunabilir ve yazılabilir özelliklerdir.

`font` özelliği ile canvas üzerine eklenecek metinler için `font` (yazı tipi) ve `size` (yazı boyutu) tanımlamaları yapabilirsiniz. Bu özelliğin varsayılan değeri `"10px sans-serif"` şeklindedir. Bu özellik, CSS font özelliği ile aynı söz dizimini ve değerleri kullanır.

Kullanımı: `context2d.font[=value]`

Örnek: `ctx.font = "italic 10px Arial";`

`textAlign` özelliği; tanımlanan noktaya göre yazının nasıl konumlandırılacağını ayarlamak için kullanılır. Bu özellik `start`, `end`, `left`, `right`, `center` değerlerinden birini alır.

- **`start:`** Varsayılan değer, canvas elemanı için tanımlanan yazı yönü (`dir` özelliği ile) soldan sağa doğru ise metnin sol kenarı metot tarafından tanımlanan noktaya (canvas üzerinde yazıya başlangıç noktası) hizalanır. Eğer canvas ele-

manı için tanımlanan yazı yönü (`dir` özelliği ile) sağdan sola doğru ise metinin sağ kenarı metot tarafından tanımlanan noktaya (canvas üzerinde yazıya başlangıç noktası) hizalanır.

- **`end:`** Canvas elemanı için tanımlanan yazı yönü (`dir` özelliği ile) soldan sağa doğru ise metinin sağ kenarı metot tarafından tanımlanan noktaya (canvas üzerinde yazıya başlangıç noktası) hizalanır. Eğer canvas elemanı için tanımlanan yazı yönü (`dir` özelliği ile) sağdan sola doğru ise metinin sol kenarı metot tarafından tanımlanan noktaya (canvas üzerinde yazıya başlangıç noktası) hizalanır.
- **`left:`** Metnin sol kenarı metot tarafından tanımlanan başlangıç noktasına hizalanır.
- **`right:`** Metnin sağ kenarı metot tarafından tanımlanan başlangıç noktasına hizalanır.
- **`center:`** Metnin orta noktası metot tarafından tanımlanan başlangıç noktasına konumlandırılır.

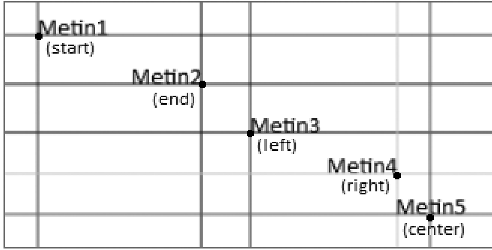
Kullanımı: `context2d.textAlign[=value]`

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>textAlign()</title>
    <style>
    canvas
    {
        border:1px solid gray;
    }
    </style>
    <script type="text/javascript">
        var renkler = ["red", "blue", "maroon", "#9DDAF5", "#F516BD"];
        var draw = function () {
            var ctx =
document.getElementsByTagName ("canvas")[0].getContext("2d");
            ctx.font = "15px calibri";
            // Metin1 textAlign:start(Varsayılan)
            ctx.fillText("Metin1", 20, 20);
            ciz(ctx, 20, 20);
```

```
            // Metin2 textAlign:end
            ctx.textAlign = "end";
            ctx.fillText("Metin2", 120, 50);
            ciz(ctx, 120, 50);
            // Metin3 textAlign:left
            ctx.textAlign = "left";
            ctx.fillText("Metin3",150, 80);
            ciz(ctx, 150, 80);
            // Metin4 textAlign:right
            ctx.textAlign = "right";
            ctx.fillText("Metin4", 240, 105);
            ciz(ctx, 240, 105);
            // Metin5 textAlign:center
            ctx.textAlign = "center";
            ctx.fillText("Metin5", 260, 130);
            ciz(ctx, 260, 130);
        }
        var i = 0;
        var ciz = function (ctx, x, y) {
            /*Buradaki fonksiyon Metinlerin canvas üzerine hangi noktadan başlanarak
              yerleştirildiğini (Başlangıç noktasını) görsel olarak görebilmeniz için çizgiler
              oluşturuyor.*/
            ctx.strokeStyle = renkler[i];
            ctx.beginPath();
            ctx.moveTo(x, 0);
            ctx.lineTo(x, 150);
            ctx.moveTo(0, y);
            ctx.lineTo(300, y);
            ctx.stroke();
            i++;
        }
    </script>
</head>
<body onload="draw();">
       <canvas id="canvas" width="300" height="150">
              Tarayıcı canvas elemanını desteklemiyor.
       </canvas>
</body>
</html>
```

Görsel olarak sonuca bakalım.



Yukarıdaki metotta başlangıç noktalarına göre metnin nasıl hizalandığı gösterilmektedir. Siyah çember içerisine alınan noktalar yazı metotlarının kullandığı başlangıç noktalarıdır.

`textBaseline` özelliği ile satır taban çizgilerinin nasıl hizalanacağını ayarlayabilirsiniz. Aldığı değerler: `top`, `hanging`, `middle`, `alphabetic`, `ideographic`, `bottom`'dır. Varsayılan değer, `alphabetic`'dir.

fillText() ve strokeText()

- `fillText()` metodu; `fillStyle` özelliği ile tanımlanan rengi kullanarak ve tanımlanan başlangıç noktasını referans alarak canvas üzerine metin yazdırmaya yarar.
- `syrokeText()` metodu; `strokeStyle` özelliği ile tanımlanan rengi kullanarak ve tanımlanan başlangıç noktasını referans alarak canvas üzerine metin yazdırmaya yarar.

  `fillText()` metodu metni dolgu çizimi ile oluştururken `strokeText()` metodu metni kenarlık çizimi ile oluşturur.

Kullanımları:

```
context2d.fillText(text,x,y [,maxWidth])
context2d.strokeText(text,x,y [,maxWidth])
```

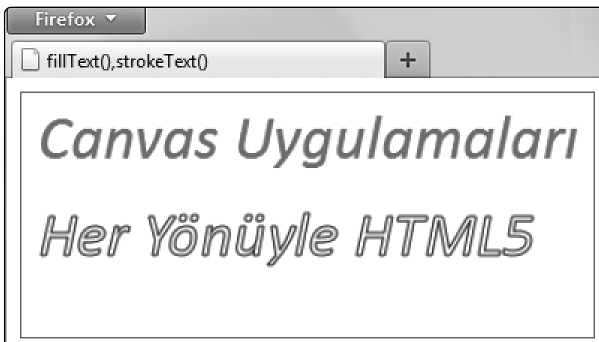
`text` parametresi; yazdırılacak metni, `x`, `y` parametreleri; metin yazdırılırken referans alınacak başlangıç noktasını ayarlar. İsteğe bağlı olan `maxWidth` parametresi ile metin için maksimum bir genişlik tanımlayabilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="utf-8">
    <title>fillText(),strpkeText()</title>
    <style>
    canvas
    {
        border:1px solid gray;
    }
    </style>
    <script type="text/javascript">
        var renkler = ["red", "blue", "maroon", "#9DDAF5", "#F516BD"];
        var draw = function () {
            var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
            ctx.font = "italic 200 38px/4 calibri";
            ctx.fillStyle = "red";
            ctx.strokeStyle = "blue";
            ctx.fillText("Canvas Uygulamaları", 10, 40);
            ctx.strokeText("Her Yönüyle HTML5",10,100);
        }
    </script>
</head>
<body onload="draw();">
      <canvas id="canvas" width="350" height="150">
            Tarayıcı canvas elemanını desteklemiyor.
      </canvas>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

MEASURETEXT()

Bu metot içerisinde tanımlanan metnin genişliğini saklayan `canvasTextMetrics` nesnesi döndürür. Bu metot içerisinde tanımlanan metnin genişliği font özelliğine atanan değerler baz alınarak bulunur.

Kullanımı:

```
metrics=context2d.measureText(text)
metrics.width
```

Örnek:

```
<!DOCTYPE html>
<html>
<head>
     <meta charset="utf-8">
     <title>measureText()</title>
     <style>
          canvas
          {
               border: 1px solid gray;
          }
     </style>
     <script type="text/javascript">
          var init = function () {
               var ctx =
document.querySelector("#canvas").getContext("2d");
               ctx.fillStyle = "#54FF05";
               ctx.strokeStyle = "rgb(0,0,255)";
               ctx.font = "italic bold 12px/2 calibri";
               /*font özelliğinin aldığı değerler ve söz dizimi CSS font özelliği ile aynıdır.
                 font:[font-style||font-variant||font-weight]?font-size[/line-height]? font-family
                 */
               var ctMetrics = ctx.measureText("Javascript Patterns !");
               console.log(ctMetrics.width);
               // Sonuç:100px
          }
     </script>
</head>
<body onload="init();">
     <canvas id="canvas" width="300" height="100">
```

```
</canvas>
</body>
</html>
```

Resimlerle Çalışmak

Canvas elemanı üzerine harici bir resim dosyasını çizdirebilirsiniz. `Context2d` nesnesi resim işlemleri için `drawImage()` metodunu tanımlar (Bu metodu kullanarak canvas elemanı üzerine bir video elemanını ya da başka bir canvas elemanı içeriğini çizdirebilirsiniz.)

drawImage()

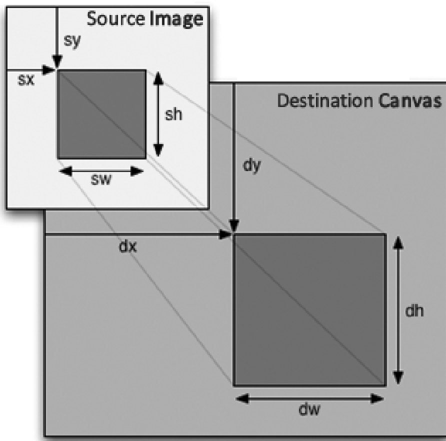
`context2d` nesnesi üzerine harici bir resmi çizdirmek için kullanılır. Bu metot; `3`, `5` ve `9` parametreli olarak kullanılabilir. Aşağıda kullanım şekilleri ve parametrelerinin açıklamaları bulunmaktadır.

```
context2d.drawImage(image,dx,dy)
context2d.drawImage(image,dx,dy,dw,dh)
context2d .drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh)
```

`(dx, dy)` parametreleri; canvas üzerinde resmin sol köşesinin yerleştirileceği noktayı ayarlar. `(dw, dh)` parametreleri; canvas üzerine yerleştirilecek resmin genişliğini ve yüksekliğini ayarlamak için kullanılır.

`(sx, sy)` parametreleri ile kaynak resimden alınacak bölgenin sol üst köşesi tanımlanırken, `(sw,sh)` parametreleri ile kaynak resimden alınacak bölgenin genişliği ve yüksekliği ayarlanır.

Metot parametrelerinin daha iyi anlaşılması için görsel olarak gösterelim.



Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>drawImage()</title>
    <script type="text/javascript">
        var init = function () {
            var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
            var img = new Image();
            img.src = "seman.png";
            /*Yeni bir img nesnesi oluşturduk ve src özelliği ile kaynak resmi tanımladık*/
            img.onload = function () {
                /*Firefox,IE'de sorun çıkarmaması adına drawImage() metodunu  canvas
                  üzerine çizdireceğimiz resmin onload olayında kullanmamız  gerekir.*/
                ctx.drawImage(img, 50, 20, 145, 120);
                // context2d.drawImage(image,dx,dy,dw,dh)
            }
            ctx.strokeStyle = "#D8CFFF";
            ctx.lineWidth = 1;
            for (var y = 0; y <= 150; y += 10) {
                ctx.beginPath();
                ctx.moveTo(0, y);
                ctx.lineTo(300, y);
                ctx.stroke();
            }
            for (var x = 0; x <= 300; x += 10) {
                ctx.beginPath();
                ctx.moveTo(x, 0);
                ctx.lineTo(x, 150);
                ctx.stroke();
            }
        }
    </script>
</head>
<body onload="init();">
    <canvas id="canvas" width="300" height="150">
    </canvas>
</body>
</html>
```

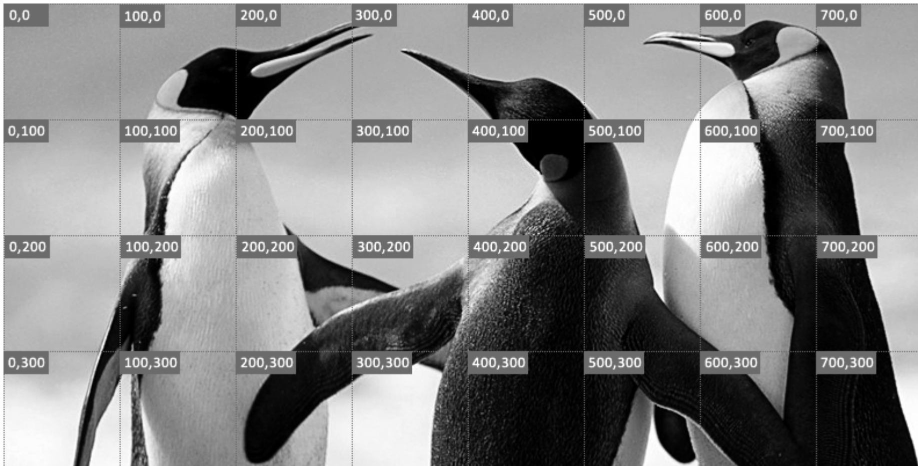
Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Diğer bir örneğimize geçelim bu örnek ile aslında bir puzzle oyununun temellerini atmış olacağız. `(800*400) px` boyutlarında bir resmimimiz olsun. Bu resim üzerinde `32` tane nokta tanımlıyoruz. Aslında bu noktalar resmi `32` eşit parçaya bölmemizi sağlayacak.

Aşağıdaki resimde gösterilen koordinatlar içinde bulundukları karenin sol üst köşesinin koordinatlarıdır. Aslında biz resmi gerçekte parçalamış felan değiliz. Fakat çalışma anında `drawImage()` metodu ile aşağıdaki resim parçalarından herhangi birini alıp canvas üzerindeki herhangi bir bölüme çizdireceğiz. Aşağıdaki resmi inceleyiniz.



Aslında yapılan işlem `drawImage()` metoduyla alınabilecek 32 eşit resim parçası tanımlanmaktır. Resmi 32 eşit parça olarak düşündüğümüzden dolayı canvas elemanınıda 32 eşit parçaya bölünmüş gibi düşüneceğiz. Resim ve canvas elemanlarının boyutları aynı olacak ve bu elemanların ikisinide 32 eşit parça şeklinde düşüneceğiz. Her bir parçanın sol üst köşesinin koordinatları resimdeki koordinatlar olacaktır.

İlk önce bu koordinatları saklayan bir dizi oluşturalım.

```
var cor = [];
for (var y = 0; y <= 300; y += 100) {
     for (var x = 0; x <= 700; x += 100) {
          cor.push([x, y, x + 100, y + 100]);
     }
}
```

Bu dizi içerisine varsaydığımız 32 eşit parçanın sol-üst ve sağ-alt noktalarının kordinatlarını atayalım. Dizinin durumu aşağıda görünmektedir.

```
[[0, 0, 100, 100], [100, 0, 200, 100], [200, 0, 300, 100], [300, 0, 400,
100], [400, 0, 500, 100], [500, 0, 600, 100], [600, 0, 700, 100], ...
```

Yukarıdaki dizi içerisindeki alt dizilerin sadece `0.` ve `1.` elemanları kullanılacak. Yaptığımız işlem ile canvas ve resim üzerinde 32 nokta tanımladık. Aslında bu noktaların oluşan karelerin sol üst köşelerini tanımladıklarını düşünürsek 32 eşit alan tanımladık diyebiliriz. Bu dizide ilk parçanın (Aşağıda konum olarak söz edeceğiz) index numarası 0 olacaktır.

`0-31` değerlerini içeren iki tane dizi tanımlıyoruz.

```
var corSrc = [];
var corDes = [];

for (var i = 0; i < 32; i++) {
     corSrc.push(i);
     corDes.push(i);
}
```

Şimdi algoritmayı nasıl kuracağımıza bakalım.

Rasgele iki tane sayı üretilmesini sağlayacağız ve bu sayıların birincisi resim üzerinden `drawImage()` metoduyla alınacak parçanın konumu tanımlayacak diğeri ise canvas üzerinde resmin çizileceği konumu tanımlayacak. Bu bulunan sayıların `corSrc`,

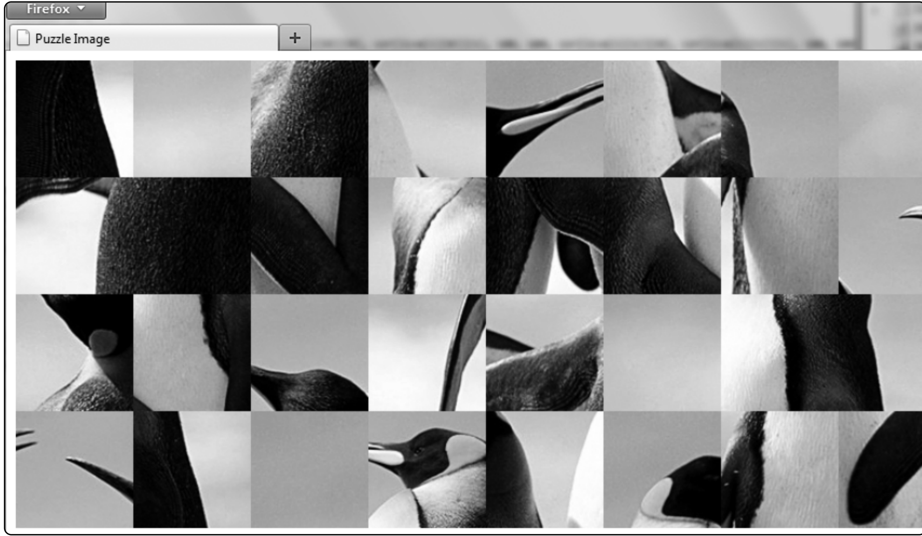
corDes dizilerinde olup olmadığına bakacağız. Eğer yoksa bu sayıları daha önce kullandığımızdan yeni rasgele iki sayı üretilmesini sağlayacağız. Eğer bu sayılar belirtilen dizilerde varsa bu sayıları live[] isimli diziye atayıp belirtilen sayıları corSrc, corDes dizilerinden sileceğiz (0-31 arasında tekrar aynı sayıyı kullanmamak için). Sonuç olarak bu sayıları index numarası olarak kullanıp cor[] dizisinden iki tane konum elde etmiş olacağız. Her defasında bulunan bu çift değerler live[] dizisine alt dizi olarak eklenecek. Ve sonuç olarak live[] dizisi içerisine drawImage() metodunun kullanacağı 32 farklı (sx, sy), (dx, dy) değeri atanmış olacak.

Kodun tamamı şöyle olacaktır:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Puzzle Image</title>
<script type="text/javascript">
    var cor = [];
    var corSrc = [];
    var corDes = [];
    var img;
    var ctx;
    var init = function () {
        ctx = document.getElementsByTagName("canvas")[0].getContext("2d");
        for (var y = 0; y <= 300; y += 100) {
            for (var x = 0; x <= 700; x += 100) {
                cor.push([x, y, x + 100, y + 100]);
            }
        }
        for (var i = 0; i < 32; i++) {
            corSrc.push(i);
            corDes.push(i);
        }
        random();
    }
    var live = [];
    var putImage = function () {
        for (var i = 0; i < 32; i++) {
            ctx.drawImage(document.getElementById("res"),
```

```
                cor[live[i][0]][0], cor[live[i][0]][1], 100, 100,
                cor[live[i][1]][0], cor[live[i][1]][1], 100, 100);
                /*
                context2d .drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh)
                */
            }
        }
        var random = function () {
            while (true) {
/*
Bu döngü live[] dizisi içerisinde 32 tane 2 elemanlı farklı alt dizi oluşuncaya kadar devam edecektir.
*/
                if (live.length == 32) {
                    putImage();
                    return;
                }
                var rnd = Math.round(Math.random() * 31); // Source
                var rnd_ = Math.round(Math.random() * 31);   // Destination
                if ((corSrc.indexOf(rnd) !== -1) &&
(corDes.indexOf(rnd_) !== -1)) {
                    live.push([rnd, rnd_]);
                    corSrc.splice(corSrc.indexOf(rnd), 1);
                    corDes.splice(corDes.indexOf(rnd_), 1);
                }
            }
        }
</script>
</head>
<body onload="init();">
<canvas id="cvs" width="800" height="400">
       Tarayıcı canvas elemanını desteklemiyor.
</canvas>
       <img id="res" src="parca.png" style="visibility: hidden;" />
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Sonuç olarak sayfa her yüklendiğinde 32 eşit resim parçasının canvas üzerinde farklı alanlara çizildiğini görmüş olacaksınız. Bu örneğe bir sonraki başlıkta tekrar döneceğiz.

Pixel Manipulation

Canvas elemanı üzerindeki piksel renk bilgilerine byte seviyesinde ulaşabilir ve bu değerlerle işlemler yapabilirsiniz.

createImageData()

Belirtilen boyutlarda bir `ImageData` nesnesi oluşturmak için kullanılır. `ImageData` nesnesi, varsayılan olarak şeffaf siyah renkli oluşturulur. **Piksel düzenleme** (*pixel manipulation*) işlemlerinde renk **RGBA** formatında kullanılır.

Kullanımı: `imagedata = context2d.createImageData(sw, sh)`

`sw`, `sh` parametreleri oluşturulacak `ImageData` nesnesinin genişliği ve yüksekliğini ayarlar.

`ImageData` nesnesi aşağıdaki özelliklere sahiptir.

Özellik	Açıklama
data	ImageData nesnesinin tanımlandığı alandaki piksellerin RGBA formatında renk değerlerini içeren CanvasPixelArray tipinde bir dizi döndürür.
height	ImageData nesnesinin px cinsinden yüksekliği.
width	ImageData nesnesinin px cinsinden genişliği.

Konunun daha iyi anlaşılması için `4px*2 px` bir `ImageData` nesnesinin pixel bilgilerinin `CanvasPixelArray` dizisi içerisinde hangi sırayla ve nasıl saklandığına bakalım.

Pixel 0				Pixel 1				Pixel 2				Pixel 3			
0 Red	1 Green	2 Blue	3 Alpha	4 Red	5 Green	6 Blue	7 Alpha	8 Red	9 Green	10 Blue	11 Alpha	12 Red	13 Green	14 Blue	15 Alpha
16 Red	17 Green	18 Blue	19 Alpha	20 Red	21 Green	22 Blue	23 Alpha	24 Red	25 Green	26 Blue	27 Alpha	28 Red	29 Green	30 Blue	31 Alpha
Pixel 4				Pixel 5				Pixel 6				Pixel 7			

Yukarıdaki şekilde, kanallar (*red, green, blue, alpha*) içerisine yazılan sayısal değerler belirtilen kanalın sakladığı değerin `CanvasPixelArray` dizisi içerisindeki yerini (index numarasını) gösterir.

Dikkat edilirse `ImageData` nesnesinin sol üst köşesinden başlamak kaydıyla her bir pikselin renk değeri `4` kanal ile ifade edilmektedir. Yani `CanvasPixelArray` dizisinin ilk `4` değeri ilk pikselin renk değerini tanımlar. Dizi içerisinde `0` indeks numaralı elemandan başlamak kaydıyla her bir `4` değer bir pikselin renk değerini tanımlar. Her bir pikselin renk değeri `RGBA()` formatında tanımlanır. Her bir renk kanalı alpha kanalı dahil `0-255` arasında bir değer alabilir.

> **NOT** `CanvasPixelArray` dizisinin `length` özelliğini kullanarak dizi içerisindeki eleman sayısını öğrenebilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>createImageData()</title>
<script type="text/javascript">
```

```
    var init = function () {
        var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
        var imageData = ctx.createImageData(60, 20);
        console.log(imageData.data);
        // Sonuç: [CanvasPixelArray] [0,0,0,0,0,...]
        console.log(imageData.data.length);
        // Sonuç:4800
    }
</script>
</head>
<body onload="init();">
<canvas id="cvs" width="400" height="200">
Tarayıcı canvas elemanını desteklemiyor.
</canvas>
</body>
</html>
```

`createImageData()` metodunun belirtilen boyutlarda şeffaf siyah renkte bir `ImageData` nesnesi oluşturduğunu daha önce söylemiştik. `[CanvasPixelArray]` dizisine bakarsak;

```
                     0.pixel 1.pixel 2.pixel
[CanvasPixelArray]  [0,0,0,0,0,0,0,0,0,0,0,0,....................]
```

Buradaki ilk dört değer sol üst köşedeki `0` index numaralı ilk pikselin renk değerini tanımlar. Her bir pixel için yapılan `0,0,0,0 rgba` tanımlaması şeffaf siyah rengi temsil eder.

> **NOT** Oluşturulan `ImageData` nesnesinin boyutu `60*20` idi. Bu durumda bu nesne içinde `1200` tane pixel bulunacaktır. Her bir pikselin renk değeri dizi içerisinde `4` sayı ile temsil edildiğinden `CanvasPixelArray` dizisinin eleman sayısı `4800` olur.

getImageData ()

Canvas üzerinde; referans olarak tanımlanan dikdörtgen içinde bulunan piksellerin sahip oldukları renk değerlerini saklayan bir `ImageData` nesnesi elde etmek için kullanılır.

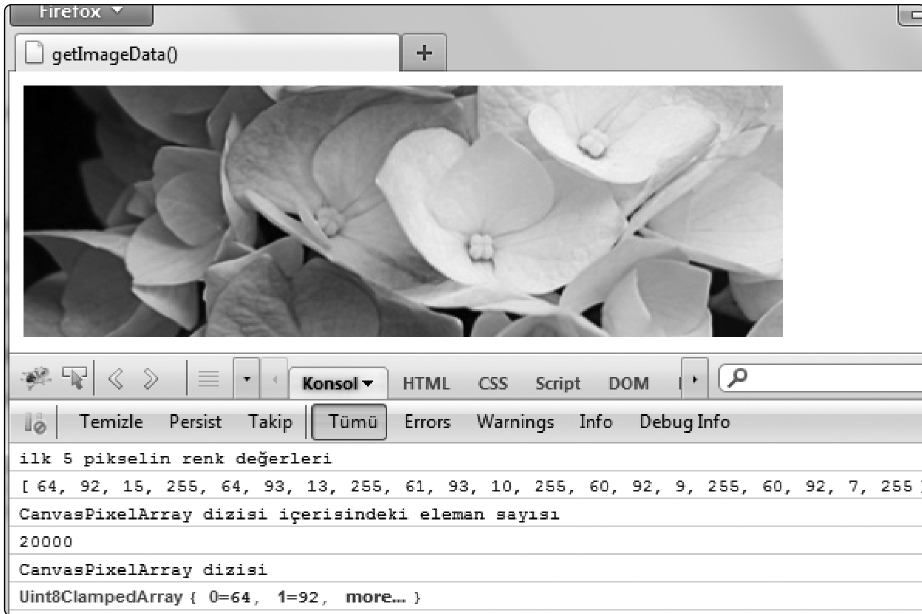
Kullanımı: `imagedata = context2d.getImageData(sx, sy, sw, sh)`

`(sx,sy)` parametreleri ile referans dikdörtgenin sol üst köşesi, `(sw,sh)` parametreleri ile dikdörtgenin (alanın) genişliği ve yüksekliğini tanımlanır. Burada yanlış anlamayın. Bu metot bir dikdörtgen çizmez. Sadece tanımlanan dikdörtgenin içinde kalan piksellerin renk bilgilerini saklayan `ImageData` nesnesi oluşturur.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>getImageData()</title>
<script type="text/javascript">
    var dizi = [];
    var init = function () {
        var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
        var img = new Image();
        img.src = "get.jpg";
        img.onload = function () {
            ctx.drawImage(img, 0, 0);
            var _gImageData = ctx.getImageData(100, 0, 100, 50);
            for (var i = 0; i < 20; i++) {
                /*ImageData nesnesinin data özelliği geriye CanvasPixelArray tipinde
                  bir dizi döndürüyordu. Bu dizinin ilk 20 elemanını yani ilk 5  pikselinin
                  renk değerlerini çıktı olarak göstermek için bir başka dizi içerisine
                  atıyoruz*/
                dizi.push(_gImageData.data[i]);
            }
            console.log("ilk 5 pikselin renk değerleri");
            console.log(dizi);
            console.log("CanvasPixelArray dizisi içerisindeki
eleman sayısı");
            console.log(_gImageData.data.length);
            console.log("CanvasPixelArray dizisi");
            console.log(_gImageData.data);
        }
    }
</script>
</head>
```

```
<body onload=”init();”>
<canvas id=”cvs” width=”450” height=”150”>
Tarayıcı canvas elemanını desteklemiyor.
</canvas>
</body>
</html>
```



Firefox 4 ekran görüntüsü

putImageData()

Bir `ImageData` nesnesini canvas üzerine çizdirmek için kullanılan metodtur. `ImageData` nesnesi `createImageData()` metoduyla oluşturulmuş ya da `getImageData()` metoduyla canvas üzerinden elde edilmiş olabilir.

Kullanımı: `context2d.putImageData(imagedata,dx,dy)`

`imageData` parametresi ile `ImageData` nesnesi tanımlanır. `(dx, dy)` parametreleri, `ImageData` nesnesinin sol köşesinin yerleştirileceği canvas üzerindeki noktayı tanımlar. Yukarıdaki parametrelerin yanında isteğe bağlı olarak `dirtyX`, `dirtyY`, `dirtyWidth`, `dirtyHeight` parametreleri de kullanılabilir.

Fakat bu parametreler Firefox tarafından şu an için desteklenmemektedir.

Örnek:

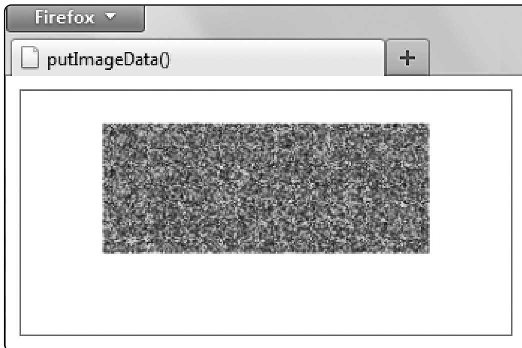
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>putImageData()</title>
<style type="text/css">
       canvas{
         {border:1px solid gray;
       }
</style>

<script type="text/javascript">
     var dizi = [];
     var init = function () {
          var ctx =
document.getElementsByTagName ("canvas")[0].getContext("2d");
          // İlk önce 100*50 boyutunda bir ImageData nesnesi oluşturalım;
          var IData = ctx.createImageData(200, 80);
          /*Yukarıda oluşturduğumuz IData nesnesinde 200*80 =16000 tane piksel
            olacaktır. Aynı zamanda her pikselin renk değeri 4 tamsayı ile tanımlandığından
            IData.data(CanvasPixelArray dizisi) 64000 elemanlı olacaktır*/
          for (var i = 0; i < IData.data.length; i += 4) {
               /*Oluşturduğum bu döngü ile ImageData nesnesinin data özelliği ile
                 tanımlanan dizinin  her  4 elemanına ayrı ayrı ulaşıp bu elemanlar için
                 (son elemana  255 tanımlaması yapmak kaydıyla) 0-255 arasında rasgele
                 bir sayı tanımlaması yapacağız. Belirtilen dizi içerisindeki her 4 elemanının
                 değeri bir pikselin rengini tanımladığından sonuçta rasgele bir renk değeri
                 elde etmiş olurum.*/
                 /*Red Kanalı*/
               IData.data[i] = Math.round(255 * Math.random());
               // Blue kanalı
               IData.data[i + 1] = Math.round(255 * Math.random());
               // Green kanalı
               IData.data[i + 2] = Math.round(255 * Math.random());
               // Alpha kanalı(Bu kanal için değer sürekli 255 olacak, şeffaflık yok)
               IData.data[i + 3] = 255;
          }
```

```
        ctx.putImageData(IData,50,20);
    }
</script>
</head>
<body onload="init();">
<canvas id="cvs" width="300" height="150">
Tarayıcı canvas elemanını desteklemiyor.
</canvas>
</body>
</html>
```

Yukarıdaki örnekte `200*80 px` boyutlarında bir `ImageData` nesnesi oluşturduk ve bu nesnenin her pikseli için rasgele bir renk tanımlaması yaptık ve daha sonra bu nesneyi `putImageData()` metoduyla canvas üzerine çizdirdik. Sonuçta canvas üzerinde farklı renklerdeki piksellerin birleşmesi ile oluşan `200*80 px` boyutlarında bir alan elde etmiş oluruz.

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Resimlerle Çalışmak isimli bölümde bir puzzle oyununun temelini atmıştık. Şimdi yazacağımız kodlarla kullanıcının karışık bir şekilde yüklenen bu resim parçalarının konumunu değiştirebilmesini sağlayalım.

Kodlarımızın en son durumunu hatırlayalım...

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="utf-8">
<title>Puzzle Image</title>
<script type="text/javascript">
    var cor = [];
    var corSrc = [];
    var corDes = [];
    var img;
    var ctx;
    var init = function () {
        ctx = document.getElementsByTagName("canvas")[0].getContext("2d");
        for (var y = 0; y <= 300; y += 100) {
            for (var x = 0; x <= 700; x += 100) {
                cor.push([x, y, x + 100, y + 100]);
            }
        }
        for (var i = 0; i < 32; i++) {
            corSrc.push(i);
            corDes.push(i);
        }
        random();
    }
    var live = [];
    var putImage = function () {
        for (var i = 0; i < 32; i++) {
            ctx.drawImage(document.getElementById("res"),
cor[live[i][0]][0], cor[live[i][0]][1], 100, 100, cor[live[i][1]][0],
cor[live[i][1]][1], 100, 100);
        }
    }
    var random = function () {
        while (true) {
            if (live.length == 32) {
                putImage();
                return;
            }
            var rnd = Math.round(Math.random() * 31);
            var rnd_ = Math.round(Math.random() * 31);
            if ((corSrc.indexOf(rnd) !== -1)
                && (corDes.indexOf(rnd_) !== -1)) {
```

```
                    live.push([rnd, rnd_]);
                    corSrc.splice(corSrc.indexOf(rnd), 1);
                    corDes.splice(corDes.indexOf(rnd_), 1);
               }
          }
     }
</script>
</head>
<body onload="init();">
<canvas id="cvs" width="800" height="400">
Tarayıcı canvas elemanını desteklemiyor.
</canvas>
<img id="res" src="parca.png" style="visibility: hidden;" />
</body>
</html>
```

Şimdi yeni kodları yazmaya başlayalım...

Kullanıcı Mouse ile konumu değiştirilecek iki resme sırasıyla tıkladığında, bu resimlerin konumu değiştirilecektir. İlk önce yapmamız gereken kullanıcının Mouse'un sol tuşu ile hangi resimler üzerinde tıkladığını bulmaktır (Mouse işaretçisinin konumunu bulmak).

İlk önce canvas elemanının `click` olayı için bir olay dinleyicisi ekleyelim; (`init()` fonksiyonu içerisine)

```
document.querySelector("canvas").addEventListener("click",mClick,false);
```

Canvas üzerinde kullanıcı Mouse'un sol tuşuna bastığında `mClick` isimli fonksiyon çalışacak.

`mClick` fonksiyonu ve diğer değişkenler;

```
     var ch = [];
     var durum = 0;
     var mClick = function (event) {
          durum++;
          var evt = event || window.event;
          var mPx = evt.clientX - document.querySelector("canvas").offsetLeft;
          var mPy = evt.clientY - document.querySelector("canvas").offsetTop;
          for (var i = 0; i < cor.length; i++) {
```

```
/*Mouse işaretçisinin hangi resim parçası üzerinde olduğunu bulmak için for döngüsü oluşturuldu.
cor[] dizisi içerisindeki alt dizilerde bulunan ilk iki elemanı daha önce kullanmıştık. cor[] dizisi
içerisindeki alt dizilerin sakladığı değerlerin ilk ikisi tanımlanan karenin sol üst köşesinin diğer
ikisi de karenin  sağ alt köşesinin koordinatlarını saklar.*/
            if (mPx > cor[i][0] && mPx <= cor[i][2]
&& mPy > cor[i][1] && mPy <= cor[i][3]) {
                ch.push(cor[i][0], cor[i][1]);
            }
        }
        switch (durum) {
            case 1:
                ctx.strokeRect(ch[0], ch[1], 100, 100);
                break;
            case 2:
                ctx.strokeRect(ch[2], ch[3], 100, 100);
                var IData = ctx.getImageData(ch[0], ch[1], 100, 100);
                var IData_ = ctx.getImageData(ch[2], ch[3], 100, 100);
                ctx.putImageData(IData, ch[2], ch[3]);
                ctx.putImageData(IData_, ch[0], ch[1]);
                durum = 0;
                ch = [];
                break;
        }
    }
```

Yukarıdaki kodlarıda ekledikten sonra artık puzzle oyununu tamamlamış duruma geldik.

Şimdi basit bir paint uygulması yapalım. Kullanıcı, canvas üzerine Mouse ile dikdörtgen, çamber, çizgi çizebilsin ve canvas üzerinde silme işlemi yapabilsin.

Uygulamayı genel olarak şöyle özetleyelim...

`Body` elemanının çocuğu durumunda olan `div#paint` elemanı içerisine bir canvas elemanı yerleştirdik. Burada tanımlanan orijinal canvas elemanıdır (Neden orijinal dediğimizi biraz sonra daha iyi anlayacaksınız). Bu orijinal canvas elemanının dışında kullanıcının çizimlerini yapabilmesi için programatik olarak bir canvas elemanı daha oluşturduk (*buffer canvas*). Programatik olarak oluşturduğumuz bu canvas elemanını orijinal canvas elemanının tam üzerine gelecek şekilde konumlandırdık (Şunu hatırlamanızı isterim: Bir canvas elemanı oluşturulduğunda varsayılan rengi siyah ve şeffaf

olacaktır). Bu canvas elemanının üzerine yapılan çizimler mouseup olayında orijinal canvas elemanı üzerine çizdirilecek ve aslında genel mantığımız bu olacak.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Basit Paint Uygulaması</title>
<style type="text/css">
    div#paint
    {
        position: relative;
        width: 800px;
        height: 400px;
        border: 1px solid gray;
    }
    #canvas2
    {
        position: absolute;
        left: 0px;
        top: 0px;
    }
    div#arac
    {
        position: absolute;
        top: 0px;
        right: -32px;
        background-color: #d6d6d6;
        width: 30px;
        height: 402px;
        text-align: center;
    }
    div#arac img
    {
        margin-top:5px;
    }
        div#arac #maske
    {
        width:30px;
        height:30px;
        position:absolute;
```

```
            background-color:crimson;
            opacity:0.6;
            top:0px;
            left:0px;
        }
    </style>
    <script type=”text/javascript”>
        var canvas1, canvas2, paint;
        var ctx1, ctx2;
        var sx, sy, ex, ey;
        var durum = false;
    /*canvas üzerinde mousedown olayının gerçekleştiğini tespit edebilmek için durum
    değişkenini kullanacağız*/
        var  maske;
        var arac = “line”;
    /*Kullanıcı canvas üzerine çizim yapmak için line, rect, ellipse araçlarından birisini kullanabilir.
    Sayfa yüklendiğinde varsayılan araç line aracıdır*/
        var init = function () {
            paint = document.querySelector(“div#paint”);
            canvas1 = document.getElementsByTagName(“canvas”)[0];
            ctx1 = canvas1.getContext(“2d”);
            canvas2 = document.createElement(“canvas”);
            ctx2 = canvas2.getContext(“2d”);
            canvas2.id = “canvas2”;
            canvas2.width = canvas1.width;
            canvas2.height = canvas1.height;
            paint.appendChild(canvas2);
    /*canvas2 adıyla yeni bir canvas elemanı (buffer canvas) oluşturuldu ve paint referanslı
    elemanının içerisine appendChild() metoduyla eklendi*/
            canvas2.addEventListener(“mousedown”, mDown, false);
            canvas2.addEventListener(“mousemove”, mMove, false);
            canvas2.addEventListener(“mouseup”, mUp, false);
            document.getElementsByTagName(“img”)[0].addEventListener(“click”,
    tool, false);
            document.getElementsByTagName(“img”)[1].addEventListener(“click”,
    tool, false);
            document.getElementsByTagName(“img”)[2].addEventListener(“click”,
    tool, false);
            document.getElementsByTagName(“img”)[3].addEventListener(“click”,
    tool, false);
```

```
          maske = document.getElementById("maske");
     }
     var mDown = function (event) {
          durum = true;
/*Kullanıcının çizim işlemine başlaması ve devam etmesi için mousedown olayının
gerçekleşmesi/gerçekleşiyor olması gerekir. mousedown olayının oluştuğunu durum değişkeni
ile anlayacağız*/
          var evt = event || window.event;
          sx = evt.clientX - paint.offsetLeft;
          sy = evt.clientY - paint.offsetTop;
/*canvas elemanı üzerinde mousedown olayı gerçekleştiğinde mouse işaretçisinin bulunduğu
pozisyon bilgisini elde ettik */
     }
     var mMove = function (event) {
/*Eğer mousedown olayından sonra tuşu basılı tutup fareyi hareket ettirirsek, bu durumda
seçilen araca göre canvas üzerine çizim işlemi gerçekleştirilir. Çizim işlemi canvas2 elemanı
(buffer canvas) üzerine yapılmaktadır.*/
          if (durum) {
               var evt = event || window.event;
               ex = evt.clientX - paint.offsetLeft;
               ey = evt.clientY - paint.offsetTop;
/*Her mousemove olayında fare işaretçisinin konumu tekrar tespit ediliyor.*/
               ctx2.clearRect(0, 0, canvas2.width, canvas2.height);
/*Aşağıdaki araçlar her mousemove olayında çalışır. Bu durumda örneğin; bir dikdörtgen
çizeceksiniz, Mouse'u her hareket ettirdiğinizde canvas üzerine sol üst köşesi sabit (sx,sy) yeni
bir dikdörtgen çizilir.İşte bu durumun önüne geçmek ve sadece mouse tuşuna bastığınız nokta
ile mouse tuşunu bıraktığınız nokta arasında bir dikdörtgen çizmek için yukarıdaki clearRect()
metodunun her mousemove olayı gerçekleştiğinde çalışması gerekir. Aslında bu metot buffer
canvas elemanını her mousemove olayında temizler.*/
               switch (arac) {
                    case "line":
                         ctx2.beginPath();
                         ctx2.moveTo(sx, sy);
                         ctx2.lineTo(ex, ey);
                         ctx2.stroke();
                         break;
                    case "rect":
                         ctx2.fillStyle = "khaki";
                         ctx2.fillRect(sx, sy, ex - sx, ey - sy);
                         break;
                    case "ellipse":
```

```
                    ctx2.beginPath();
                    ctx2.arc(sx, sy, Math.abs(sx - ex), 0, 2 *
Math.PI, true);
                    ctx2.fillStyle = “green”;
                    ctx2.fill();
                     break;
            }
        }
    }
    var mUp = function (event) {
        durum = false;
        ctx1.drawImage(canvas2, 0, 0);
        ctx2.clearRect(0, 0, canvas2.width, canvas2.height);
```

/*Mouse tuşunu serbest bıraktığınızda, yani mouseup olayı gerçekleştiğinde; drawImage() metoduyla ctx2 içerisinde oluşan çizim ctx1 üzerine çizdiriliyor ve tampon çizim alanı (ctx2) clearRect() metoduyla temizleniyor.*/

```
    }
    var tool = function (event) {
        var evt = event || window.event;
        var tool_ = evt.target;
        arac = tool_.alt;
        if (arac == “delete”) {
            ctx1.clearRect(0, 0, 800, 400);
```

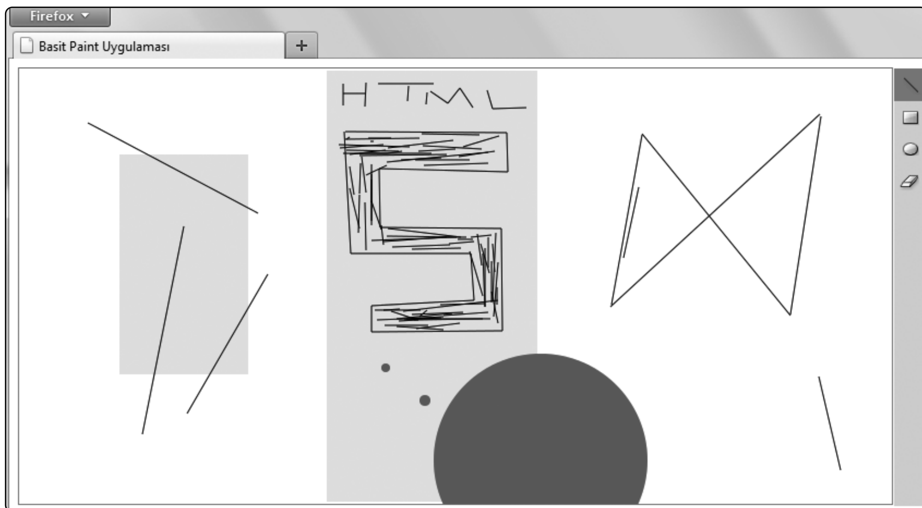
/*Kullanıcı delete tuşuna bastığında canvas1 üzerindeki tüm grafikler silinecektir.*/

```
        }
        switch (arac) {
            case “line”:
                maske.style.top = “0px”;
                break;
            case “rect”:
                maske.style.top = “30px”;

                break;
            case “ellipse”:
                maske.style.top = “60px”;
                break;
            case “delete”:
                maske.style.top = “90px”;
                break;
        }
```

```
    }
</script>
</head>
<body onload="init();">
       <div id="paint">
       <canvas id="cvs" width="800" height="400">
       Tarayıcı canvas elemanını desteklemiyor.
       </canvas>
       <div id="arac">
              <img src="line.png" alt="line" />
              <img src="rec.png" alt="rect" />
              <img src="el.png" alt="ellipse" />
              <img src="sil.png" alt="delete" />
              <div id="maske"></div>
       </div>
       </div>
</body>
</html>
```

Ekran görüntüsüne bakalım:



Firefox 4 ekran görüntüsü

Örnek: Bu örneğimizde "gülen yüz" isimli basit bir oyun tasarlayacağız. Kodlarımızı yazarak içerisinde açıklamaları yapalım.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>oyun_</title>
<style type="text/css">
    canvas#buffer
    {
        position: absolute;
        left:8px;
        top:8px;
    }
    input[type="text"]
    {
        background-color:khaki;
        border:none;
        font-family:Calibri;
        font-size:16px;
    }
    a
    {
        background-color:#bbccff;
        border:none;
        font-family:Calibri;
        font-size:16px;
        text-decoration:none;
     }
</style>
<script type="text/javascript">
/*Oyunumuzda ana canvas elemanımızın  dışında sadece 1 tane buffer canvas elemanını
programatik olarak oluşturup kullanacağız. Ana canvas ve buffer canvas elemanlarımız
420*240 boyutlarında olacak. Ana canvas elemanı üzerine programatik olarak oluşturduğumuz
buffer canvas elemanını yerleştireceğiz. Özet olarak işlem adımları;

1. İlk olarak ana canvas ve buffer canvas elemanlarında resim yerleştirirken ve mouse click
olayında kullanacağımız koordinat bilgilerini saklayacak bir dizi oluşturacağız. Bu dizi ile
beraber biz canvas elemanını canvas(0,0) noktasından başlayarak 112 eşit (30px*30px)
```

parçaya bölünmüş gibi düşüneceğiz ve bu dizi içerisinde bu eşit parçaların(karelerin) sol üst ve sağ alt noktalarının koordinatlarını saklayacağız. Bu dizi içerisinde 112 alt dizi bulunacak. Ana canvas elemanı üzerine sml.png,nosml.png resimlerini rasgele yerleştirirken ve kullanıcının buffer canvas üzerinde hangi kareye tıkladığını bulmak için cort[] isimli diziyi kullanacağız.

2. Ana canvas elemanı üzerinde rasgele konumlara , 40 tane nosml.png ve 72 tane sml.png resmi yerleştireceğiz.Sayfa her yüklendiğinde yukarıdaki resimlerin yerleri değişecek. Dolayısıyla 112 resmin her defasında rasgele yerleştirildiği bir canvas elemanımız olmuş olacak.Peki bu işlemi nasıl yaptık diyeceksiniz ?

konum[] , live[] dizi değişkenleri ve konumHesapla() fonksiyonu bu işlem için oluşturuldu.

3. Programatik olarak oluşturduğumuz buffer canvas elemanı üzerine beyaz renkli bir dikdörtgen ve buffer canvas üzerinde 30*30 büyüklüğünde kareler oluşturacak şekilde bir ızgara çizdirdik. Artık ana canvas üzerine çizdirdiğimiz resimler buffer canvas elemanı ile maskelenmiş oldu. (sml.png,nosml.png resmileri gözükmüyor)

4. Kullanıcının mouse ile tıkladığı konumda bulunan ana canvas elemanı üzerindeki resmi göstermemiz gerekecek. Bu durumda eğer kullanıcının tıkladığı karede nosml.png resmi yüklü ise oyun bitirilecek yani buffer canvas elemanı üzerindeki tüm renk değerleri temizlenecek (Buffer canvas üzerine clearRect() metoduyla siyah şeffaf bir dikdörtgen çizilecek), Eğer kullanıcının tıkladığı karede sml.png resmi yüklü ise buffer canvas elemanı üzerinde hangi kare içerisine tıkladığını bulup bu kare üzerindeki renk değerlerini temizliyoruz (clearRect() metoduyla siyah şeffaf bir kare çizmiş olacağız). Dolayısıyla sadece kullanıcının tıkladığı kare altındaki sml.png resmini kullanıcıya göstermiş olacağız.

5.Kullanıcının oyun başladıktan sonra kaç tane sml.png resmi açtırdığını öğrenmesi için (Oyun puanını) bir tane text elemanı oluşturduk ve yukarıdaki mClick() fonksiyonu ile ilişkilendirdik.

Evet sonuç olarak genel mantığımız bu olmakla beraber size karışık gibi gelebilir. Aslında javascript ve canvas kullanılarak yapılan oyun tasarımlarında iyi bir algoritma kurmanız şarttır. Yani sürükle bırak programcılığı burada yoktur. Ne kadar iyi temel oluşturursanız, uygulamanızda o kadar sade ve okunabilir olacaktır. Yukarıda uygulamanın genel hatları anlatılmıştır. Kodlara baktığınızda bir bilgiyi elde etmek için çok değişik varyasyonlar kullandığımızı göreceksiniz. Kod satırları daha iyi anlamanız için bazı yerlerde bilinçli olarak uzatılmıştır .Aşağıdaki kodu yazmanız ve her satırın ne iş yaptığı ile ilgili verdiğim bilgiler ışığında düşünmenizi tavsiye ederim.

/*Global değişkenlerimizi tanımlayalım.*/

```
var cort = [];
```

/*cort[] isimli dizi değişkeni içerisinde canvas elemanının sol üst köşesinden başlamak kaydıyla 112 tane eşit karenin koordinatlarını tanımlayacağız. cort[] isimli dizi içerisine bakarsak;

cort=[[0,0,30,30],[30,0,60,30],…]

Buradan dizi içerisinde 112 tane alt dizi olduğunu ve bu alt diziler içerisinde karenin sol-üst köşesi ve sağ-alt köşesinin saklandığını görebilirsiniz.

[x,y,x+30,y+30] x,y değerleri karelerin sol-üst köşesinin koordinatlarıdır.*/

```
var konum = [];
```

/*konum isimli dizi içerisinde [0,1,2,3,…,111] şeklinde sayıları saklıyoruz.

Bu diziyi ne için kullanacağız?

konumHesapla() isimli fonksiyonu 112 defa çalıştırıp 112 tane değişik sayı elde edeceğiz (Bu sayılar live[] isimli dizi içerisinde saklanacak) ve bu sayıları index numarası olarak kullanıp her defasında cort[] dizisi içerisinden 112 tane değişik konum bilgisi elde etmiş olacağız. cort[] dizisi içerisindeki bir konumu (cort[] alt dizisini/elemanını) sadece 1 defa kullanmak için konum[] isimli diziye ihtiyacımız var. Yani şöyle diyelim random(), round() fonksiyonları yardımıyla ile 0<=x<112 arasından bir x sayısı elde edince bu sayıyı konum[] isimli dizi içerisinden silip bu sayının bir daha kullanılmasını engelliyoruz.*/

```
  var cBuffer, ctxBuffer;
  var live = [];
```

/*live[] dizisi konumHesapla() isimli fonksiyonun oluşturduğu 112 değişik sayıyı saklamak için kullanılacak*/

```
  var tikCor, tikResult;
```

/*tikCor değişkenini kullanıcının tıkladığı karenin tanımlı olduğu cort[] dizi içerisindeki alt dizinin index numarasını saklamak için, tikResult değişkenini ise kullanıcının tıkladığı karenin altına ana canvas elemanında belirtilen konumda çizili olan resim bilgisini elde etmek için kullanacağız.*/

```
  var say = 0;
```

/*Puan bilgisini ekrana yazdırmak için say değişkeni oluşturdum*/

```
  var oyunDurum = true;
```

/*Oyun durum bilgisini elde etmek için oyunDurum değişkenini oluşturdum.*/

```
for (var y = 0; y <= 210; y += 30) {
        for (x = 0; x <= 390; x += 30) {
            cort.push([x, y, x + 30, y + 30]);
        }
    }
    for (var i = 0; i < cort.length; i++) {
        konum.push(i);
    }
var init = function () {
        var ctx =
document.getElementsByTagName("canvas")[0].getContext("2d");
```

/*Oyunda kullanacağım iki tane resim nesnesi oluşturdum ve src özelliği(property) ile bu nesnelere kaynak tanımlaması yaptım*/

```
        var img = new Image();
        img.src = "sml.png";
        var img_ = new Image();
        img_.src = "nosml.png";
```

/*cBuffer isimli canvas elemanını createElement() metoduyla oluşturdum ve gerekli tanımlamaları yaptım*/

```
        cBuffer = document.createElement("canvas");
        document.getElementById("txt").value = 0;
        cBuffer.id = "buffer";
        cBuffer.width = ctx.canvas.width;
        cBuffer.height = ctx.canvas.height;
        document.body.appendChild(cBuffer);
        ctxBuffer = cBuffer.getContext("2d");
        ctxBuffer.strokeStyle = "gray";
        ctxBuffer.fillStyle = "white";
        ctxBuffer.fillRect(0, 0, 420, 240);
        // cBuffer elemanı için click olay dinleyicisi ekledim
        cBuffer.addEventListener("click", mClick, false);
        /*cBuffer elemanı üzerinde [30px*30px] karelerini oluşturacak
          ızgarayı oluşturalım */
        for (var x = 0; x <= 420; x += 30) {
            ctxBuffer.beginPath();
            ctxBuffer.moveTo(x, 0);
            ctxBuffer.lineTo(x, 240);
            ctxBuffer.stroke();
        }
        for (var y = 0; y <= 240; y += 30) {
            ctxBuffer.beginPath();
            ctxBuffer.moveTo(0, y);
            ctxBuffer.lineTo(420, y);
            ctxBuffer.stroke();
        }
/*İki resmin onload olaylarının sonucunda  ana canvas elemanı üzerine toplam 112 tane resim
yerleştirmiş olacağım*/
        img.onload = function () {
/*konumHesapla() fonksiyonunu 112 kere çalıştırılarak live[] dizisi içerisinde 0 ile 111 arasından
farklı 112 tane değer elde etmiş olurum*/
            for (var i = 0; i < 112; i++) {
                konumHesapla();
            }
            for (var i = 0; i < 72; i++) {
/*live[] dizisi içerisindeki ilk 72 değeri index numaraları olarak kullanıp cort[] dizisinden elde
edeceğim 72 farklı konuma sml.png resmini çizdireceğim.*/
                ctx.drawImage(img, cort[live[i]][0], cort[live[i]][1]);
            }
        }
```

```
        img_.onload = function () {
/*live[] dizisi içerisindeki son 40 değeri index numaraları olarak kullanıp cort[] dizisinden elde
edeceğim 40 farklı konuma nosml.png resmini çizdireceğim.*/
            for (var i = 72; i < 112; i++) {
                ctx.drawImage(img_, cort[live[i]][0], cort[live[i]][1]);
            }
        }
    }
    var konumHesapla = function () {
/*Bu fonksiyon 112 defa çalıştırılacak ve sayfa her yüklendiğinde live[] dizisi içerisinde farklı
olarak sıralanmış 0 ile 111 arasından değerler saklanmış olacak*/
        while (true) {
            var rnd_ = Math.round(Math.random() * 111);
            if (konum.indexOf(rnd_) != -1) {
                live.push(rnd_);
                konum.splice(konum.indexOf(rnd_), 1);
                return ;
            }
        }
    }
/*konumHesapla() isimli fonksiyonun görevini daha iyi anlamanızı sağlayalım; Örneğin
konumHesapla() isimli fonksiyonun ilk çalışması; Bu durumda while() döngüsü içerisindeki
rnd_ değişkeninin 15 değerini sakladığını düşünelim 15 değeri konum[] isimli dizide aranacak
eğer var ise (15 değerinin daha önce kullanılmadığı anlamına gelir) bu değer live dizisine
eklenir ve konum dizisinden silinir.
live=[15],
konum=[0,1,2,3,…,14,16,,,,111], değerlerini alırlar ve  return deyimi ile döngüden çıkılır.
Bu fonksiyunu 2. çalışmasına bakalım;
rnd_ değişkeninin tekrar 15 değeri elde ettiğini düşünelim bu durumda bu değer artık konum[]
dizisi içerisinde olmayacağından while() döngüsü ile tekrar rasgele bir sayı elde edilerek
yukarıdaki adımlar tekrar edilir. İşte böylece konumHesapla() isimli fonksiyon her çağrıldığında
live[] dizisi içerisine 0-111 arasından farklı bir sayı eklenir.*/
   var mClick = function (event) {
        if (!oyunDurum) {
            return;
        }
        var event = event || window.event;
        var mouseX = event.clientX - cBuffer.offsetLeft;
        var mouseY = event.clientY - cBuffer.offsetTop;
/*mouseClick olayı gerçekleştiğindeki Mouse işaretçisinin konumunu elde ediyoruz.*/
```

```
          for (var i = 0; i < cort.length; i++) {
/*click olayı gerçekleştiğinde, Mouse işaretçisinin canvas üzerinde tanımlı hangi kare üzerine
olduğunu bulmak için bu döngü oluşturuldu. tikCor değişkeni kullanıcının tıkladığı karenin
tanımlı olduğu cort[] dizi içerisindeki alt dizinin index numarasını elde etmemizi sağlar.*/
               if (mouseX > cort[i][0] && mouseX <= cort[i][2] &&
mouseY > cort[i][1] && mouseY <= cort[i][3]) {
                    tikCor = i;
               }
          }
/*Yukarıda mouse işaretçisinin bulunduğu cort[] dizisi içerisindeki konum bilgisinin index
numarasını buldum. Şimdi, daha önce söylemiştim sayfa her yüklendiğinde live[] dizisi içerisinde
0 ile 111 dahil arasındaki sayılar rasgele sıralanıyordu. Bu sayılardan ilk 72 sinin kullanımıyla
cort[] dizisinden elde edilen konumlara sml.png son 40'nın kullanımı ile cort[] dizisinden elde
edilen konumlara nosml.png resmi çizdiriliyordu. O zaman live[] dizisi içerisindeki ilk 72 sayı
gülen yüzü son 40 sayı ise ağlayan yüzü çizdirmek için kullandığımız cort[] index numaralarıdır.
Eğer tikCor değişkeni içerisindeki sayı 72 eşit ve küçükse bu durumda bu alanda gülen yüz
çizilidir ve oyun devam etmelidir. Eğer tikCor içerisindeki sayı 72'den büyük ise o zaman bu
alanda ağlayan yüz çizilidir ve oyun sonlanmalıdır. */
              tikResult = live.indexOf(tikCor);
          if (tikResult >= 72) {
               ctxBuffer.clearRect(0, 0, 420, 240);
               for (var x = 0; x <= 420; x += 30) {
                    ctxBuffer.beginPath();
                    ctxBuffer.moveTo(x, 0);
                    ctxBuffer.lineTo(x, 240);
                    ctxBuffer.stroke();
               }
               for (var y = 0; y <= 240; y += 30) {
                    ctxBuffer.beginPath();
                    ctxBuffer.moveTo(0, y);
                    ctxBuffer.lineTo(420, y);
                    ctxBuffer.stroke();
               }
               document.getElementById("txt").value = "Oyun Bitti
Toplam Puan:" + say;
               oyunDurum = false;

          } else {
               ctxBuffer.clearRect(cort[tikCor][0] + 1,
cort[tikCor][1] + 1, 28, 28);
               say++;
```

```
                document.getElementById("txt").value = say;
            }
        }
</script>
</head>
<body onload="init();">
<canvas id="canvas" width="420" height="240">
</canvas><br />
<input type="text" size="25" id="txt" disabled />
<a href="oyun_.html">Tekrar Oyna !</a>
</body>
</html>
```

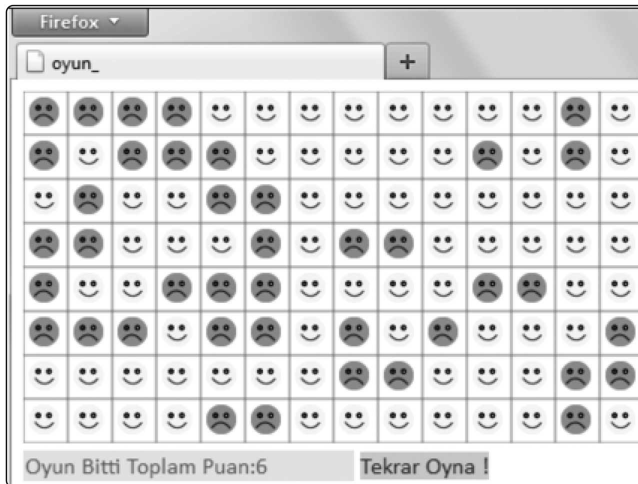
Oyunu tekrar oynamak için sayfayı tekrar yüklemeniz ya da **Tekrar Oyna!** linkine tıklamanız yeterli olacaktır. Tekrar Oyna! linki, sayfayı tekrar tarayıcıya yükleyecektir (Body elemanı için load olayını oluşturur).

Kullanılan resimler:



sml.png nosml.png

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Canvas İle Animasyon Temelleri

Canvas+JavaScript kullanarak basit ya da gelişmiş animasyonlar oluşturmanız mümkündür. HTML5 CANVAS elemanı ile animasyon oluşturmak isteyen birinin şu an için kullanabileceği herhangi bir program yoktur. Yani animasyonu oluşturacak kodların hepsi programcı tarafından düşünülerek elle yazılır. Bu durumda programcının en azından temel geometri ve matematik bilgilerine sahip olması gerekir. Canvas elemanı ile animasyon (hareketli nesneler) oluşturmak için izleyeceğimiz adımlar şu şekildedir.

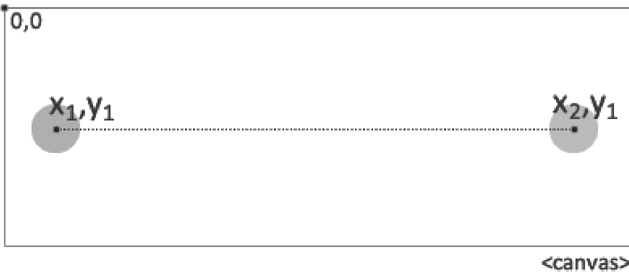
Çiz > Temizle > Güncelle

Ayrıca bazı uygulamalarda birden fazla Canvas elemanını ana Canvas elemanı üzerinde tampon eleman olarak kullanabiliyoruz. Canvas+JavaScript ile programlama yapmak bazıları için külfetli gelebilir. Ben aynı düşüncede değilim. Çünkü programlama sadece hazır programların sağladığı sürükle-bırak araçları ile yapılmamalıdır. Ayrıca HTML5 Canvas+JavaScript ile yapacağınız uygulamalar programlama gücünüzü arttıracaktır.

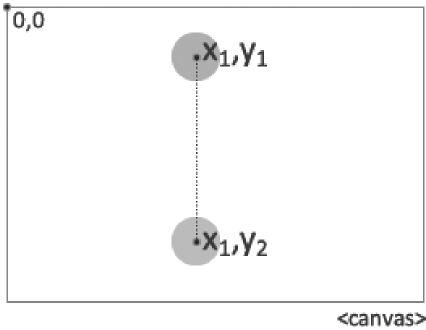
Nesnelerin Yatay ya da Dikeyde Düz Bir Çizgi Üzerinde Hareket Ettirilmesi

Nesneleri sadece bir eksende (yatayda ya da dikeyde) hareket ettirmek için nesnelerin yatay ya da dikey koordinatlarını bir zamanlayıcı ile güncelleyip belirtilen nesneyi Canvas elemanı üzerine tekrar çizdirmemiz yeterli olacaktır.

Yatay eksende hareket:



Dikey eksende hareket:

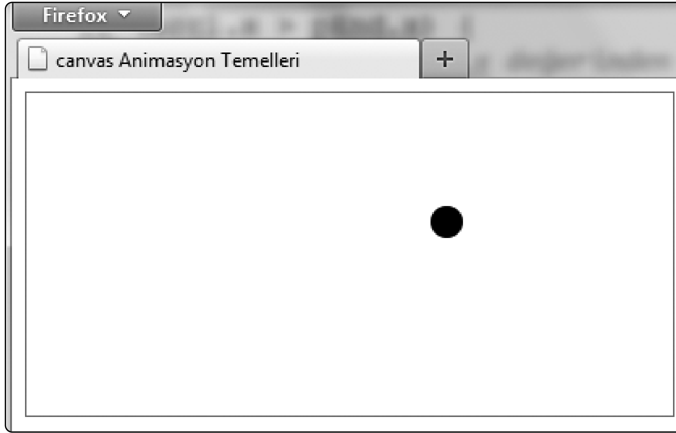


Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>canvas Animasyon Temelleri</title>
    <style type="text/css">
        canvas
        {
            border:1px solid gray;
        }
    </style>
    <script type="text/javascript">
        var ctx, arc1;
        var objectArc = function (x, y) {
            this.x = x;
            this.y = y;
        }
        /*hareket ettireceğimiz daireleri x ve y özelliklerine sahip birer nesne olarak
        tanımlayacağız. Bu işimizi epeyce kolaylaştıracaktır*/
        var pStart = {x:10,y:80};
        var pEnd = { x: 390, y: 80 };
        /*Harekete başlangıç noktasını pStart(x1,y1) ve bitiş noktasını pEnd(x2,y1) nesne
        olarak tanımlıyoruz.*/
        var play = function () {
            ctx =
document.getElementsByTagName('canvas')[0].getContext('2d');
```

```
            arc1 = new objectArc(pStart.x, pStart.y);
             /*arc1 isimli bir daire nesnesi oluşturduk*/
            draw();
        }
        var draw = function () {
            if (arc1.x > pEnd.x) {
                /*arc1.x değeri pEnd.x değerinden büyük olursa bu durumda en son
                çizimin gerçekleştiği sonucu çıkar. Yani arc metodunun çizimde
                kullandığı en son x değeri pEnd.x değeridir. Bu durumda draw()
                fonksiyonunun çalışması sonlandırılır.*/
                return;
            }
            ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
            /*Bu fonksiyon her çalıştığında clearRect() metoduyla canvas elemanı
            üzeri temizleniyor.*/
            ctx.beginPath();
            ctx.arc(arc1.x, arc1.y, 10, 0, Math.PI * 2, false);
            /*arc1 isimli daireyi canvas elemanı üzerine çizdirdik. Fonksiyon her
            çalıştığında arc1.x özelliğinin değeri değişeceğinden bu değişen değere
            göre daire tekrar çizdirilir.*/
            ctx.fill();
            arc1.x += 10;
            setTimeout('draw()', 50);// zamanlayıcı
        }
    </script>
</head>
<body onload="play();">
    <canvas id="canvas1" width="400" height="200">
    </canvas>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Yukarıdaki örnekte nesnenin `pStart` noktasından `pEnd` noktasına hareket etmesini sağladık. Şimdi nesnenin bu iki nokta (`pStart`, `pEnd` noktaları) arasında sürekli gidip gelmesini sağlayalım.

İlk önce nesnenin geriye doğru harekete geçmesini sağlamak için `drawBack()` isimli bir fonksiyon oluşturacağım bu fonksiyon `draw()` fonksiyonu içerisinden **`arc1.x > pEnd.x`** şartı gerçekleştiğinde çağrılacak (Yani daire `pEnd` nesnesinin tanımladığı koordinatlara geldikten sonra).

```
var draw = function () {
            if (arc1.x > pEnd.x) {
                drawBack();
                return;
            }
            ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
            ctx.beginPath();
            ctx.arc(arc1.x, arc1.y, 10, 0, Math.PI * 2, false);
            ctx.fill();
            arc1.x += 10;
            setTimeout('draw()', 50);
        }
```

`drawBak()` isimli fonksiyonumuzun kodları:

```
var drawBack = function () {
            if (arc1.x < pStart.x) {
            /*arc1.x özelliğinin değeri pStart.x'den küçük olursa bu durumda daire
            çiziminde kullanılan en son x değeri pStart.x şeklinde olmuştur.
            Yapmamız gereken şey nesneyi ileri(sağa) hareket ettirmek olacaktır.
            Bu yüzden draw() fonksiyonu çağrılmıştır.
            */
                draw();
                return;
            }
            ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
            ctx.beginPath();
            ctx.arc(arc1.x, arc1.y, 10, 0, Math.PI * 2, false);
            ctx.fill();
            arc1.x -= 10;
            /*Nesnenin geriye doğru hareketini sağlamak için arc1.x özelliğinin değerini
            sürekli azaltacağız.*/
            setTimeout('drawBack()', 50);
        }
```



Nesneleri dikeyde doğrusal bir çizgi üzerinde hareket ettirmenin mantığı yukarıda anlatılan kodlardan çok farklı olmayacaktır. Nesnelerin x koordinat değerlerini sabit tutarak y koordinat değerlerini bir zamanlayıcı ile değiştirirseniz, nesneleri doğrusal olarak dikey eksende hareket ettirmiş olursunuz. Aşağıdaki örnekte nesnelerin dikey hareket ettirilmesi anlatılmıştır.

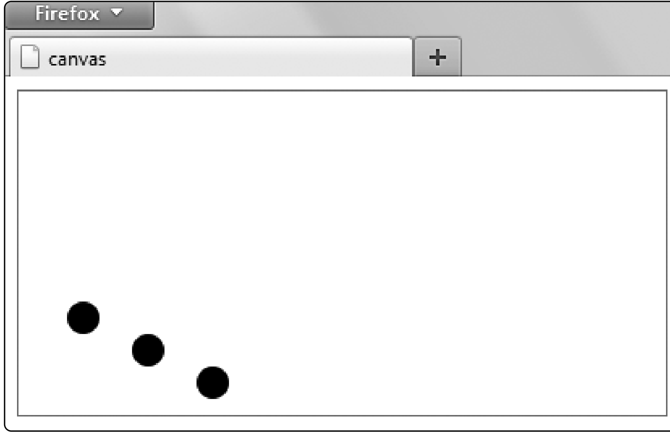
Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>canvas</title>
    <style type="text/css">
        canvas
        {
            border:1px solid gray;
        }
    </style>
```

```
     <script type=”text/javascript”>
          var ctx;
          var pointY = {y1:15,y2:35,y3:55};
/*canvas üzerine 3 tane daire çizdireceğiz. Bu dairelerin y koordinat değerlerini pointY isimli
bir nesne içerisinde saklıyoruz.*/
          var play = function () {
               ctx =
document.getElementsByTagName(‘canvas’)[0].getContext(‘2d’);
               draw();
          }
          var draw = function () {
               if (pointY.y3  == ctx.canvas.height-15) {
                    return;
                    /*3.dairenin y koordinatı 185px olunca nesnelerin hareketini
               durduruyoruz.*/

               }
               ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
               ctx.beginPath();
               ctx.arc(40, pointY.y1, 10, 0, Math.PI * 2, false);
               ctx.arc(80, pointY.y2, 10, 0, Math.PI * 2, false);
               ctx.arc(120, pointY.y3, 10, 0, Math.PI * 2, false);
               ctx.fill();
               pointY.y1 += 5;
               pointY.y2 += 5;
               pointY.y3 += 5;
               /*Dairelerin y koordinat değerleri arttırılıyor ve setTimeout ile draw()
fonksiyonunun tekrar çalıştırılması sağlanıyor.*/
               setTimeout(‘draw()’, 50);
          }
     </script>
</head>
<body onload=”play();”>
     <canvas id=”canvas1” width=”400” height=”200”>
     </canvas>
</body>
</html>
```

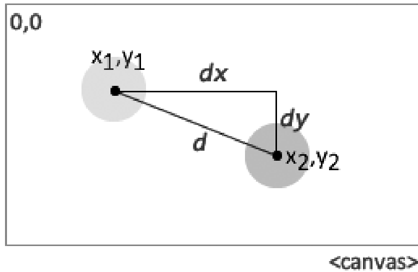
Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Nesnelerin Yatay ve Dikeyde Beraber Hareket Ettirilmesi

Bir nesnenin aynı anda hem yatay hem de dikey eksende hareket etmesini sağlayabilirsiniz. Aşağıdaki şekli inceleyiniz.



<canvas>

Yukarıdaki şekle bakarak gerekli formülleri yazalım:

$dx = x2 - x1$, $dy = y2 - y1$, $d = \sqrt{(dx)^2 + (dy)^2}$

Örnek: Bu uygulama ile yatay da (sağa ya da sola) ve dikeyde (aşağıya ya da yukarıya) rasgele `20 px` hareket eden bir daire oluşturacağız ve çizdireceğimiz daire, canvas elemanı içerisinden çıkmayacak.

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="utf-8" />
    <title>canvas</title>
    <style type="text/css">
        canvas
        {
            border: 1px solid gray;
        }
    </style>
    <script type="text/javascript">
        var ctx ,x,y;
        /*Daireyi nesne olarak  oluşturacağımızdan dolayyı arcObject isimli bir kurucu
        fonksiyon tanımlanmıştır.
        (constructor Pattern)
        */
        var arcObject = function (x, y, r) {
            this.x = x;
            this.y = y;
            this.r = r;
        }
        var arc = new arcObject(20, 30, 10);
        /*arcObject isimli kurucu fonksiyon kullanılarak arc isimli bir nesne oluşturuldu*/
        var play = function () {
            ctx =
document.getElementsByTagName('canvas')[0].getContext('2d');
            draw();
        }
        var draw = function () {
            ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
            ctx.beginPath();
            ctx.arc(arc.x, arc.y, arc.r, 0, Math.PI * 2, false);
            ctx.fill();
            /*arc nesnesinin ilk x,y değerleri ile tanımlanan noktada bir daire çizdirildi*/
            while (true) {
                x = Math.round(Math.random()) * 40 - 20;
                y = Math.round(Math.random()) * 40 - 20;
                if (!(arc.x + x < arc.r || arc.x + x
+ arc.r > ctx.canvas.width || arc.y + y < arc.r || arc.y + y
+ arc.r > ctx.canvas.height)) {
                    break;
                }
```

```
            }
            arc.x += x;
            arc.y += y;
            setTimeout('play()', 50);
        }
    </script>
</head>
<body onload="play();">
    <canvas id="canvas1" width="400" height="200">
    </canvas>
</body>
</html>
```

Yukarıda uygulamada daire ilk olarak Canvas elemanı üzerine (x=20, y=30) merkezli ve 10 px yarıçaplı olarak çizdirilecektir. Daha sonra amacımız dairenin merkez koordinatlarını (x,y) 20 px arttırarak ya da azaltarak daireyi tekrar çizdirmek olacaktır. Böylece daire rasgele yatayda sola ya da sağa, dikeyde üste ya da alta 20 px hareket edecektir. Fakat dairenin her defasında Canvas elemanı içerisine çizdirilmesi önemlidir. (Canvas elemanı içinde kalması) daireyi her çizimde Canvas elemanı içerisinde tutmak için aşağıdaki `while()` döngüsü oluşturulmuştur.

```
while (true) {
                x = Math.round(Math.random()) * 40 - 20;
                y = Math.round(Math.random()) * 40 - 20;
                if (!(arc.x + x < arc.r || arc.x + x + arc.r >
ctx.canvas.width || arc.y + y < arc.r || arc.y + y + arc.r >
ctx.canvas.height)) {
                    break;
             }
}
```

`While` döngü parantezleri içerisine `true` değerini yazıyoruz. Aslında bu durumda sonsuz bir döngü oluşturmuş olduk. Fakat istediğimiz şartlar oluşunda döngüden `break` deyimini kullanarak çıkmamız mümkün. Döngü içerisinde **`x`** ve **`y`** değişkenleri +20 ya da -20 değerlerinden birini rasgele alacaktır.

`Math.round(Math.random())` ifadesi geriye 1 ya da 0 değerini döndürür. Eğer 1 değerini döndürürse örneğin; x=20, eğer 0 değerini döndürürse x=-20 olacaktır. Bulunan bu x, y değerleri dairenin merkez koordinatlarına ekleyeceğimiz değerler olacaktır (Dairenin yeni merkez noktası `arc.x=arc.x+x, arc.y=arc.y+y` şeklinde olacak).

`if()` yapısı ile yeni çizilecek dairenin Canvas elemanı sınırları içerisinde kalıp kalmayacağı kontrol ediliyor. Eğer yeni değerlerle daire Canvas elemanı içerisine çizdirilebiliyor ise, break deyimi ile döngüden çıkmamız gerekecektir. Eğer yeni değerlerle daire Canvas elemanı dışına çizdirilecek ise, bu durumda döngü tekrar çalışacak ve yeni `x`, `y` değerleri hesaplayacaktır. Döngü, daireyi Canvas elemanı içerisinde tutacak; `x` ve `y` değerleri bulununcaya kadar devam eder.

Peki, dairenin her defasında Canvas elemanı içerisine çizilmesini nasıl sağlıyoruz?

`if()` yapısı içerisindeki şartlara yakından bakalım.

```
arc.x + x < arc.r
```

Eğer yeni çizilecek dairenin x koordinatı (arc.x+x) dairenin yarı çapından küçük olursa, bu durumda daire Canvas elemanının sol köşesinden dışarı çıkar.

```
arc.x + x + arc.r >  ctx.canvas.width
```

Eğer yeni çizilecek dairenin x koordinatı `(arc.x + x)` ve dairenin yarı çapının toplamı canvas elemanının genişliğinden büyük olursa daire canvas elemanının sağ köşesinden dışarı çıkar.

```
arc.y + y < arc.r
```

Eğer yeni çizilecek dairenin y koordinatı `(arc.y + y)` dairenin yarı çapından küçük olursa bu, durumda daire Canvas elemanının üst köşesinden dışarı çıkar.

```
arc.y + y + arc.r > ctx.canvas.height
```

Eğer yeni çizilecek dairenin y koordinatı `(arc.y + y)` ve dairenin yarı çapının toplamı Canvas elemanının yüksekliğinden büyük olursa daire Canvas elemanının alt köşesinden dışarı çıkar.

Son olarak daireyi Canvas elemanı içerisinde tutan `x`, `y` değerlerini dairenin merkez koordinatlarına ekleyip zamanlayıcıyı yazıyoruz.

```
arc.x += x;
arc.y += y;
setTimeout('play()', 50);
```

Yukarıdaki uygulamada daire Canvas elemanı içerisine rasgele çizdiriliyordu. Gelin dairenin Canvas üzerinde çizdirildiği yerin boyanmasını sağlayalım.

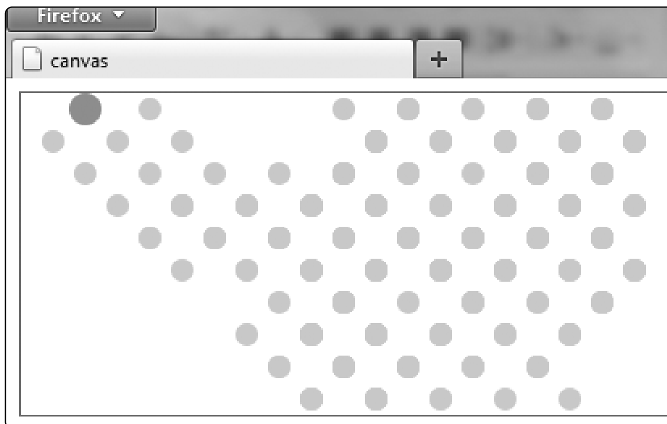
```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="utf-8" />
    <title>canvas</title>
    <style type="text/css">
        canvas
        {
            border: 1px solid gray;
        }
    </style>
    <script type="text/javascript">
        var ctx1,ctx2,x,y;
        var arcObject = function (x, y, r) {
            this.x = x;
            this.y = y;
            this.r = r;
        }
        var arc = new arcObject(20, 30, 10);
        var play = function () {
            ctx1 =
document.getElementsByTagName('canvas')[0].getContext('2d');
            ctx2 =
document.getElementsByTagName('canvas')[1].getContext('2d');
            ctx1.fillStyle = "#ffcc00";
            ctx2.fillStyle = "#00bb00";
            draw();
        }
        var draw = function () {
            ctx2.clearRect(0, 0, ctx2.canvas.width, ctx2.canvas.height);
            ctx2.beginPath();
            ctx2.arc(arc.x, arc.y, arc.r, 0, Math.PI * 2, false);
            ctx2.fill();
            ctx1.beginPath();
            ctx1.arc(arc.x, arc.y, arc.r-3, 0, Math.PI * 2, false);
            ctx1.fill();

            while (true) {
                x = Math.round(Math.random()) * 40 - 20;
                y = Math.round(Math.random()) * 40 - 20;
                if (!(arc.x + x < arc.r || arc.x + x + arc.r >
ctx2.canvas.width || arc.y + y < arc.r || arc.y + y + arc.r >
```

```
ctx2.canvas.height)) {
                      break;
                  }
              }
              arc.x += x;
              arc.y += y;
              setTimeout('play()', 50);
          }
    </script>
</head>
<body onload="play();">
<div style="width:400px;height:200px;position:relative;">
    <canvas id="canvas1" width="400" height="200">
    </canvas>
    <canvas style="position:absolute;left:0px;top:0px;" width="400"
height="200"></canvas>
</div>
</body>
</html>
```

Ekran görüntüsü:



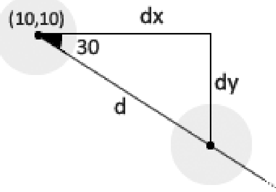
Firefox 4 ekran görüntüsü

Örnek: Bu örnekte bir nesnenin bir vektör boyunca nasıl hareket ettirildiğine bakacağız. Yani nesneyi bir açıyla hareket ettireceğiz.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>canvas</title>
    <style type="text/css">
        canvas
        {
            border: 1px solid gray;
        }
    </style>
    <script type="text/javascript">
        var d= 5;
        var pStart = {
            x: 10,
            y: 10
        };
        var angle = 30;
        var rad = (angle * Math.PI) / 180;
        var dx = Math.cos(rad) * d;
        var dy = Math.sin(rad) * d;
        var daire = pStart;
        var nokta = [];
        var draw = function () {
            var ctx =
document.getElementsByTagName('canvas')[0].getContext('2d');
            ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
            nokta.push([daire.x, daire.y]);
            ctx.fillStyle = '#cc00cc'; // küçük dairelerin dolgu rengi
            for (var i = 0; i < nokta.length; i++) {
                ctx.beginPath();
                ctx.arc(nokta[i][0], nokta[i][1], 1, 0, Math.PI * 2,
false);
                ctx.fill();
            }
            ctx.fillStyle = '#888800'; // Dairenin dolgu rengi
            ctx.beginPath();
            ctx.arc(daire.x, daire.y, 10, 0, Math.PI * 2, false);
            ctx.fill();
            daire.x += dx;
```

```
            daire.y += dy;
            setTimeout('draw()', 40);
        }
    </script>
</head>
<body onload="draw();">
    <canvas id="canvas1" width="400" height="200">
    </canvas>
</body>
</html>
```

Yukarıdaki uygulamada yapılan işlemi görselleştirerek inceleyelim.



İlk daire `(10,10)` noktası merkezli olarak çizdirilecektir. Diğer dairelerin merkez noktaları; ilk dairenin merkez noktasından `30`^{0}'lik bir açı ile geçen vektör üzerine olmalıdır (Yani daireler bir açı ile hem yatay hem de dikeyde hareket etmiş olacaklar).

Uygulamada ilk önce değişken ve nesne tanımlamaları yapılmıştır.

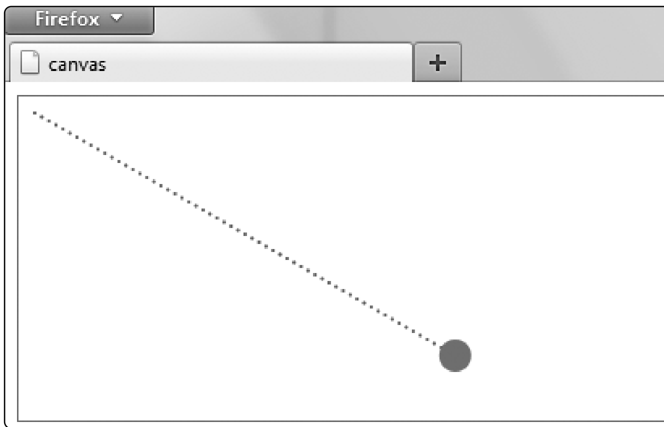
```
var d= 5;
var pStart = {
    x: 10,
    y: 10
};
var angle = 30;
var rad = (angle * Math.PI) / 180;
var dx = Math.cos(rad) * d;
var dy = Math.sin(rad) * d;
var daire = pStart;
var nokta = [];
```

Şekilden de anlaşılacağı üzere aslında `d` değişkeni ile hareket hızını ayarlayabilirsiniz. `pStart` nesnesi ilk noktanın koordinatlarını tanımlamak için oluşturulmuştur. `dx` ve `dy` değişkenleri dairenin `30`^{0}'lik bir açıyla hareket etmesi için her defasında dairenin merkez noktalarına eklenecek sabit değerleri saklamaktadır.

`draw()` fonksiyonu içerisinde hesaplanan koordinatlara daire çizdirildikten sonra, yeni çizilecek dairenin merkez koordinatları; `daire.x`, `daire.y` değerlerine `dx`, `dy` değerleri eklenerek elde edilecektir.

`draw()` fonksiyonu içerisinde tanımlanan `nokta` isimli dizi değişkenine dikkat edin. Bu dizi dairenin çizdirildiği merkez noktalarını saklamak için oluşturulmuştur. Her daire çiziminde ilk önce `nokta[]` değişkeni içerisindeki değerleri (dairenin önceden çizildiği merkez noktalarını) kullanarak küçük daireler çizdiriyoruz. Amacımız dairenin hareket yolunu görsel olarak işaretlemek.

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Yukarıdaki örnekte daireyi Canvas elemanı içerisinde tutma gibi bir isteğimiz olmadı. Şimdi bu dairenin vektörel hareketler ile daire içerisinde kalmasını sağlayalım.

Örnek:

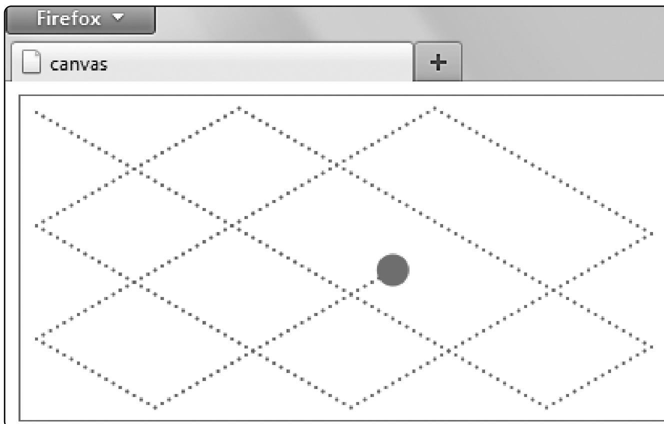
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>canvas</title>
    <style type="text/css">
        canvas
        {
            border: 1px solid gray;
```

```
        }
    </style>
    <script type="text/javascript">
        var daireCreate = function (x, y, r) {
            this.x = x;
            this.y = y;
            this.r = r;
        }
        var daire = new daireCreate(10, 10, 10);
        var d = 5;
        var angle = 30;
        var rad = angle * Math.PI / 180;
        var dx = Math.cos(rad) * d;
        var dy = Math.sin(rad) * d;
        var nokta = [];

        var radUpdate = function () {
           /*Daire canvas elemanı köşelerine geldiğinde radUpdate() fonksiyonu
            çağrılacaktır. Bu fonksiyon güncellenen açıya göre aşağıdaki değerleri
            tekrar hesaplar*/
            rad = (angle * Math.PI) / 180;
            dx = Math.cos(rad) * d;
            dy = Math.sin(rad) * d;
        }
        var draw = function () {
            var ctx =
ocument.getElementsByTagName('canvas')[0].getContext('2d');
            ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
            nokta.push([daire.x, daire.y]);
            ctx.fillStyle = '#cc00cc'; // Küçük dairelerin dolgu rengi
            for (var i = 0; i < nokta.length; i++) {
                ctx.beginPath();
                ctx.arc(nokta[i][0], nokta[i][1], 1, 0, Math.PI * 2,
false);
                ctx.fill();
            }
            ctx.fillStyle = '#888800'; // Ddairenin dolgu rengi
            ctx.beginPath();
            ctx.arc(daire.x, daire.y, daire.r, Math.PI * 2, false);
            ctx.fill();
            daire.x += dx;
```

```
            daire.y += dy;
             /*Daire canvas elemanının yatay ya da dikeyde köşelerine geldiğinde
             dairenin canvas elemanı içerisinde kalmasını sağlamak için açı değerlerini
             değiştiriyoruz. */
            if (daire.x + daire.r > ctx.canvas.width || daire.x <
daire.r) {
                angle = 180 - angle;
                radUpdate();
            } else if (daire.y + daire.r > ctx.canvas.height ||
daire.y < daire.r) {
                angle = 360 - angle;
                radUpdate();
            }
            setTimeout('draw()', 44);
        }
    </script>
</head>
<body onload="draw();">
    <canvas id="canvas1" width="400" height="200">
    </canvas>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Noktalı çizgiler, dairenin hareket yollarını göstermektedir.

Nesnelerin Dairesel Olarak Hareket Ettirilmesi

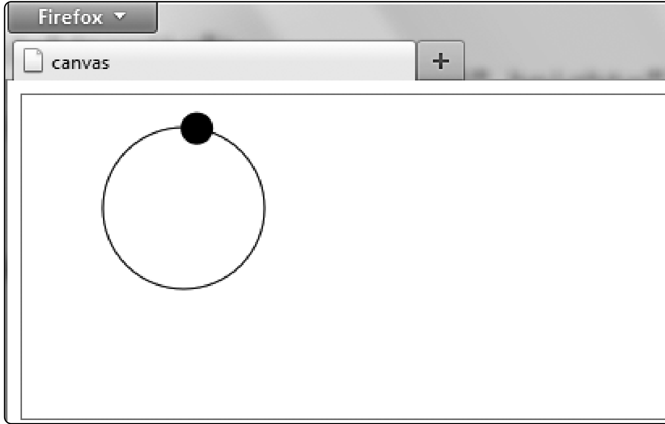
Temel geometri bilgilerini kullanarak nesneleri dairesel olarak hareket ettirmek çok basit olacaktır.

Örnek:

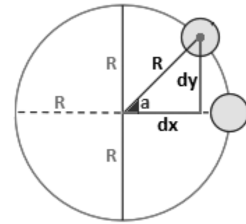
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>canvas</title>
    <style type="text/css">
        canvas
        {
            border: 1px solid gray;
        }
    </style>
    <script type="text/javascript">
        var merkezNokta = { x: 100, y: 70,radius:50 };
        var daire = {};
        var angle = 0;
        var draw = function () {
            var ctx =
document.getElementsByTagName('canvas')[0].getContext('2d');
            ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
            daire.x =
merkezNokta.x + Math.cos(angle) * merkezNokta.radius;
            daire.y =
merkezNokta.y + Math.sin(angle) * merkezNokta.radius;
            ctx.beginPath();
            ctx.arc(daire.x, daire.y, 10, Math.PI * 2, false);
            ctx.fill();
            ctx.beginPath();
            ctx.arc(merkezNokta.x, merkezNokta.y, merkezNokta.radius, 0,
Math.PI * 2, false);
            ctx.stroke();
            angle += 50;
            setTimeout('draw()', 44);
        }
```

```
    </script>
</head>
<body onload="draw();">
    <canvas id="canvas1" width="400" height="200">
    </canvas>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü



Küçük daire çemberin kenarları ile tanımlanan yol üzerinde dönmektedir. Şimdi yukarıdaki uygulamada yapılan işlemi görselleştirerek anlatalım.

Yandaki şekilden gerekli formülleri çıkartalım.

$$\text{Sin}(\alpha) = \frac{dy}{R} \qquad \text{Cos}(\alpha) = \frac{dx}{R}$$

Dairenin merkez noktasını hareketi boyunca `R` yarıçaplı çemberin kenarında tutacağız ve daireyi `a` açısıyla hareket ettireceğiz. Bu durumda dairenin merkez noktası ile çemberin merkez noktası arasındaki yataydaki uzaklık `dx=cos(a)*R`, dikeydeki uzaklık `dy= sin(a)*R` şeklinde olacaktır.

Örneğimize geriye dönersek, özet olarak yaptığımız işlem `R(merkezNokta.radius)` değerini sabit tutarak (dairenin merkezinin çemberin kenarında kalmasını sağlayarak) açı değerini değiştirmektir.

`draw()` fonksiyonu her çalıştığında aşağıdaki satırlar ile yeni çizilecek dairenin x,y koordinatları hesaplanmaktadır.

```
daire.x = merkezNokta.x + Math.cos(angle) * merkezNokta.radius;
daire.y = merkezNokta.y + Math.sin(angle) * merkezNokta.radius;
```

Daha sonra daireyi yukarıdaki değerlere göre çizdirip yeni çizim işlemi için açı değerini arttırıyoruz.

Örnek: Bu uygulamamızda yarıçapı `10 px` olan daireler kullanıp Canvas elemanı üzerine *"HTML5"* metnini yazdıracağız. Kullanıcı Mouse ile Canvas elemanı üzerinde geldiğinde bu daireler farklı yollar izleyerek (farklı açılarla) Canvas elemanının alt kenarında toplanacaklardır. Kullanıcının Mouse'u Canvas elemanı üzerine getirdiği her durumda dairelerin izledikleri yol (kullandıkları açı) farklı olacaktır. Şimdi kodlarımızın tamamını yazıp daha sonra da inceleyelim...

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>HTML5 Animasyon</title>
    <style type="text/css">
        canvas
        {
            border: 1px solid gray;
        }
    </style>
    <script type="text/javascript">
         /*Global değişkenlerimizi tanımlayalım*/
        var canvas, ctx, daireKor, timer1;
        var daireArray = [];
        var init = function () {
            canvas = document.getElementsByTagName('canvas')[0];
            ctx = canvas.getContext('2d');
            canvas.addEventListener('mouseover', mouseOver, false);
            canvas.addEventListener('mouseout', mouseOut, false);
            /*Harfleri oluşturacak dairelerin merkez koordinatlarını(x,y) daireKor isimli
            dizi değişkeni içerisinde saklayacağız.*/
            daireKor = [
            // H
```

```
[24, 35], [24, 55], [24, 75], [24, 95], [24, 115], [44, 75],
[64, 35], [64, 55], [64, 75], [64, 95], [64, 115]
// T
,[94, 35], [114, 35], [134, 35], [114, 55], [114, 75],
 [114, 95], [114, 115]
// M
,[164, 35], [164, 55], [164, 75], [164, 95], [164, 115],
 [222, 34], [222, 54], [222, 74], [222, 94], [222, 114]
,[180, 43], [195, 54], [205, 43]
// L
,[254, 35], [254, 55], [254, 75], [254, 95], [254, 115],
 [274, 115],[294, 115]
// 5
,[374, 35], [354, 35], [334, 35], [334, 55], [334, 75],
 [354, 75], [374, 75], [374, 95], [374, 115]
,[354, 115], [334, 115]
              ];

for (var i = 0; i < daireKor.length; i++) {
    daireArray.push(new daire(daireKor[i][0],
daireKor[i][1]));
}
  /*Çizim işlemi için draw()fonksiyonunu ve her daire nesnesinin dx,dy,angle
  özelliklerine değer atanması için animasyonHesapla() fonksiyonunu
  çağırıyoruz.*/
draw();
animasyonHesapla();
}

function daire(x, y) {
    this.x = x;
    this.y = y;
    this.baseX = x;
    this.baseY = y;
    this.dx;
    this.dy;
    this.angle;
}

var animasyonHesapla = function () {
    for (var i = 0; i < daireArray.length; i++) {
```

```
                var tempDaire = daireArray[i];
/*Bu döngünün her çalışmasında elde edilen daire nesnesi tempDaire isimli bir değişken
içerisinde saklanmaktadır.*/
                d = 2 + Math.round(Math.random() * 7);
                tempDaire.angle = 45 + Math.round(Math.random() * 100);
                rad = tempDaire.angle * Math.PI / 180;
                tempDaire.dx = Math.cos(rad) * d;
                tempDaire.dy = Math.sin(rad) * d;
                daireArray[i] = tempDaire;
            }
        }

        var draw = function () {

            ctx.clearRect(0, 0,canvas.width,canvas.height);
            for (var i = 0; i < daireArray.length; i++) {
                var tmpDaire = daireArray[i];
                var rGradient = ctx.createRadialGradient(tmpDaire.x,
tmpDaire.y, 2, tmpDaire.x, tmpDaire.y, 10);
                rGradient.addColorStop(“0.0”, “#FA2AB1”);
                rGradient.addColorStop(“0.25”, “green”);
                rGradient.addColorStop(“1.0”, “khaki”);
                ctx.fillStyle = rGradient;
                ctx.beginPath();
                ctx.arc(tmpDaire.x, tmpDaire.y, 10, 0, Math.PI * 2,
false);
                ctx.fill();
            }
        }

        var mouseOver = function () {
            for (var i = 0; i < daireArray.length; i++) {
                var tempDaire = daireArray[i];
                if (tempDaire.x + tempDaire.dx + 10 > ctx.canvas.width
|| tempDaire.x + tempDaire.dx < 10) {
                    tempDaire.dx = 0;
                } else if (tempDaire.y + tempDaire.dy + 10 >
ctx.canvas.height || tempDaire.y + tempDaire.dy < 10) {
                    tempDaire.dy = 0;
                }
                tempDaire.x += tempDaire.dx;
                tempDaire.y += tempDaire.dy;
```

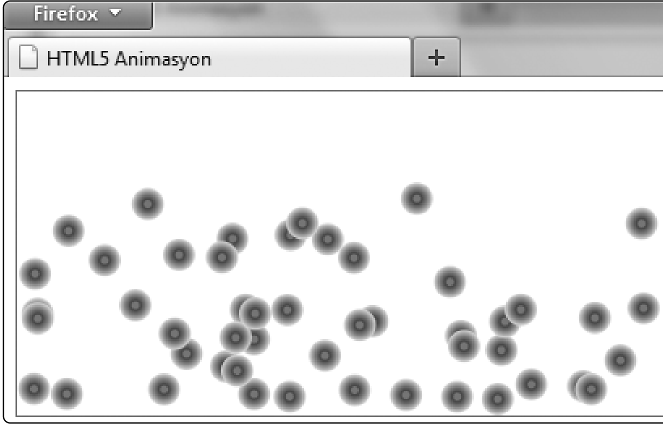
```
                daireArray[i] = tempDaire;
            }
            draw();
            timer1 = setTimeout('mouseOver()', 35);
        }
        var mouseOut = function () {
            clearTimeout(timer1);
            for (var i = 0; i < daireArray.length; i++) {
                var tempDaire = daireArray[i];
                tempDaire.x = tempDaire.baseX;
                tempDaire.y = tempDaire.baseY;
            }
            animasyonHesapla();
            draw();
        }
    </script>
</head>
<body onload='init();'>
    <canvas id="canvas1" width="400" height="200">
    </canvas>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Mouse ile Canvas elamanı üzerine geldiğinizde:



Firefox 4 ekran görüntüsü

Yazdığımızç kodları ayrıntılı bir şekilde anlatalım.

Daire Nesneleri Oluşturmak için Kurucu Fonksiyon

Uygulamamızda 49 tane daire kullandık. Bu durumda daireleri nesne olarak oluşturmak işimizi oldukça kolaylaştırır. Düşünün 49 tane daire var ve bu dairelerin değişen özellikleri, sabit özellikleri mevcut ve her daire için bu özellikler farklı durumda. Bu bakımdan sizde uygulamalarınızda kod yazarken nesnel düşünmelisiniz. Kurucu fonksiyonumuz sadece daire nesnesi oluşturmak için dışarıdan `x`, `y` (dairenin merkez noktası) koordinatlarını kabul etmektedir. Burada önemli bir ayrıntı; daire nesneleri `baseX`, `baseY` özelliklerine sahip olmalıdırlar. Çünkü biz daire nesnelerini hareket ettirdiğimizde `x`, `y` özelliklerinin aldıkları değerler değişecektir. Daireleri orijinal yerlerine geri döndürmek için `baseX`, `baseY` özelliklerine ihtiyaç duyacağız. Diğer özelliklere (`dx`, `dy`, `angle`) `animasyonHesapla()` fonksiyonu içerisinden değerler atanacaktır.

```
function daire(x, y) {
      this.x = x;
      this.y = y;
      this.baseX = x;
      this.baseY = y;
      this.dx;
      this.dy;
      this.angle;
}
```

Daire nesnelerini yukarıdaki kurcu fonksiyonu kullanarak `init()` fonksiyonu içerisinde tanımlıyoruz.

```
for (var i = 0; i < daireKor.length; i++) {
     daireArray.push(new daire(daireKor[i][0], daireKor[i][1]));
}
```

Bu daire nesneleri `daireArray` isimli dizi değişkeni içerisinde saklanacaktır. Daire nesneleri `daireKor` isimli dizi değişkeni içerisindeki koordinatlar kullanılarak oluşturulmaktadır. Örneğin; yukarıdaki döngüde `i=0` değeri için;

```
daireArray.push(new daire(daireKor[0][0],daireKor[0][1]));
```

...şeklinde bir komut satırı çalıştırılır. Bu komut satırı aşağıdaki nesneyi `daireArray` dizisi içerisine ekler.

```
new daire(24,35) >> daireArray=[Object daire(x:24,y:35)]
```

Her Bir Daire Nesnesi için Rasgele Bir Hareket Yolunun Tanımlanması (AnımasyonHesapla Fonksiyonu)

Sayfa her yüklendiğinde ya da kullanıcı Mouse'u Canvas elemanı üzerinden çektiğinde `animasyonHesapla()` fonksiyonu çalışır. Bu fonksiyon içerisinde dairelerin hareket yolları rasgele hesaplanır. Dairelerin hareket yolları `angle(açı)`, `d(hız,mesafe)` değerine bağlı olarak hesaplanır. Bu hesaplamalar her daire nesnesi için ayrı ayrı yapılmakta ve rasgele hesaplanan bu hareket yolunu temsil eden `dx`, `dy`, `angle` değerleri daire nesnelerinin ilintili özelliklerine atanmaktadır.

```
var animasyonHesapla = function () {
    for (var i = 0; i < daireArray.length; i++) {
          var tempDaire = daireArray[i];
          d = 2 + Math.round(Math.random() * 7);
          tempDaire.angle = 45 + Math.round(Math.random() * 100);
          rad = tempDaire.angle * Math.PI / 180;
          tempDaire.dx = Math.cos(rad) * d;
          tempDaire.dy = Math.sin(rad) * d;
          daireArray[i] = tempDaire;
    }
}
```

Daha önceki konularımızda bir nesnenin bir açıyla nasıl hareket ettirildiğini geniş olarak anlatmıştık. Yukarıda her bir daire yapılan işlem için; `2` ile `9` arasında rasgele bir hız (hareket miktarı: `d`) değeri `45` ile `145` arasında bir açı değeri hesaplanmak-

tadır. Bu değerlere bağlı olarak nesnemizin mevcut merkez noktası koordinatlarına eklenecek dx,dy değerleri bulunmaktadır.

Bu döngünün her çalışmasında elde edilen daire nesnesi `tempDaire` isimli bir değişken içerisinde saklanmaktadır. `For` döngüsü ile `daireArray` dizi değişkeni içerisinden elde edilen daire nesnesi için gerekli güncellemeler yapıldıktan sonra, bu nesne `daireArray` dizisi içerisindeki bulunduğu konuma tekrar yazdırılmaktadır. Yani `daireArray` dizisi belirtilen eleman için güncellenmektedir.

Daire Nesneleri için Dolgu (Dairesel Renk geçişi) Tanımlanması ve Nesnelerin Canvas Elemanı Üzerine Çizdirilmesi (Draw Fonksiyonu)

`draw()` fonksiyonu `daireArray` dizisi içerisinde saklanan daire nesnelerinin ekrana çizdirilmesi için kullanılmaktadır. Bu fonksiyonun her çalışmasında Canvas elemanı üzerini temizler. Bu fonksiyon içerisinde bir for döngüsüyle `daireArray` dizisi içerisindeki daire nesneleri Canvas elemanı üzerine çizdirilmektedir.

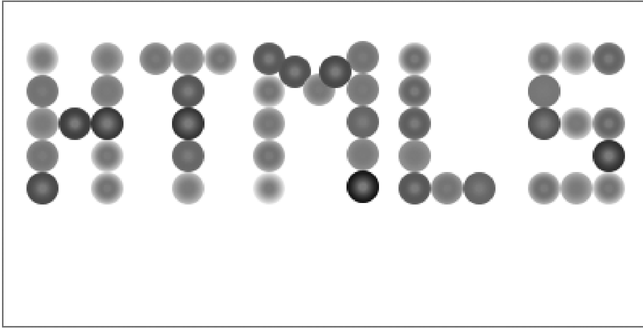
```
var draw = function () {
    ctx.clearRect(0, 0,canvas.width,canvas.height);
    for (var i = 0; i < daireArray.length; i++) {
        var tmpDaire = daireArray[i];
        var rGradient = ctx.createRadialGradient(tmpDaire.x,
tmpDaire.y, 2, tmpDaire.x, tmpDaire.y, 10);
        rGradient.addColorStop("0.0", "#FA2AB1");
        rGradient.addColorStop("0.25", "green");
        rGradient.addColorStop("1.0", "khaki");

        ctx.fillStyle = rGradient;
        ctx.beginPath();
        ctx.arc(tmpDaire.x, tmpDaire.y, 10, 0, Math.PI * 2, false);
        ctx.fill();
    }
}
```

`draw()` fonksiyonu içerisinde daire nesnelerine dolgu olarak kullanılacak bir dairesel renk geçişi nesnesi tanımlanmıştır. İsterseniz tüm daire nesneleri için bir tane dolgu tanımlayabilir ya da aşağıdaki gibi tüm daire nesneleri için rasgele hesaplanan farklı dolgular tanımlayabilirsiniz. `draw()` fonksiyonunu aşağıdaki gibi değiştirip ekran çıktısına bakalım.

```
var draw = function () {
   ctx.clearRect(0, 0,canvas.width,canvas.height);
```

```
   for (var i = 0; i < daireArray.length; i++) {
        var tmpDaire = daireArray[i];
        var rGradient = ctx.createRadialGradient(tmpDaire.x, tmpDaire.y, 2,
tmpDaire.x, tmpDaire.y, 10);
 var r = Math.round(Math.random() * 255);
        var g = Math.round(Math.random() * 255);
        var b = Math.round(Math.random() * 255);
        rGradient.addColorStop("0.0", "#FA2AB1");
        rGradient.addColorStop("1.0","rgb(" + r + "," + g + "," + b + ")");
        ctx.fillStyle = rGradient;
        ctx.beginPath();
        ctx.arc(tmpDaire.x, tmpDaire.y, 10, 0, Math.PI * 2, false);
        ctx.fill();
   }
}
```



Ekran görüntüsü Firefox 4 ile elde edilmiştir.

Canvas Elemanının mouseover Olayında Daire Nesnelerinin Rasgele Yolları Kullanarak Canvas Elemanının Alt Köşesinde Toplanması (mouseOver Fonksiyonu)

Bu fonksiyon içerisinde `for` döngüsüyle `daireArray` dizisi içerisindeki daire nesnelerine ulaşılmaktadır. Bir daire nesnesinin hareket etmesi için; daha önceden hesaplanan ve hareket yolunu tanımlayan `dx`, `dy` değerleri dairenin `x`, `y` özelliklerinin sakladıkları değerlere eklenir ve daha sonra `daireArray` dizisi içerisindeki pozisyonları güncellenen daire nesneleri `draw()` fonksiyonu tarafından Canvas elemanı üzerine çizdirilir. Dairelerin `x` özelliklerine aldıkları değerlerle (`x` özelliği dairenin hali hazırda çizili olduğu noktanın yatay koordinatıdır) `dx` özelliklerine aldıkları değerler toplanır. Yine aynı şekilde dairelerin y özelliklerine aldıkları değerlerle `dy` özelliklerinin değerleri toplanır. Amacımız; dairenin yeni çizimde Canvas elemanı sınırlarını terk

etmemesini sağlamaktır. Eğer daire yatayda ya da dikeyde Canvas elemanının köşesine gelmişse bir sonraki çalışacak mouseOver fonksiyonunun daire nesnesini Canvas elemanı içerisinden çıkarmaması için dx ya da dy değerlerine 0 değeri atanmıştır.

```
var mouseOver = function () {
            for (var i = 0; i < daireArray.length; i++) {
                var tempDaire = daireArray[i];
                if (tempDaire.x + tempDaire.dx + 10 > ctx.canvas.width
|| tempDaire.x + tempDaire.dx < 10) {
                    tempDaire.dx = 0;
                } else if (tempDaire.y + tempDaire.dy + 10 >
ctx.canvas.height || tempDaire.y + tempDaire.dy < 10) {
                    tempDaire.dy = 0;
                }
                tempDaire.x += tempDaire.dx;
                tempDaire.y += tempDaire.dy;
                daireArray[i] = tempDaire;
            }
            draw();
            timer1 = setTimeout('mouseOver()', 35);
        }
```

Özetle yukarıdaki fonksiyonda yapılan işlemler:

- Daire nesnelerine ulaşmak ve yeni çizim işlemi için daire nesnelerinin Canvas elemanı dışına çıkmasını engellemek.
- Her daire nesnesi için farklı olarak rasgele tanımlanan dx, dy değerlerini daire nesnelerinin x ve y özelliklerinin değerlerine eklemek ve dairelerin x, y özelliklerini güncellemek (Yani daireArray dizisi içerisindeki daire nesnelerinin x, y özelliklerinin değerlerini güncellemek).
- Çizim işleminin yapılması için draw() fonksiyonunu çağırmak.
- mouseOver() fonksiyonunun tekrar çalıştırılması için bir zamanlayıcı tanımlamak.

Canvas Elemanının mouseout Olayında Daire Nesnelerinin Orijinal Konumlarına Geri Dönmelerini Sağlamak (mouseOut Fonksiyonu)

Bu fonksiyon içerisinde ilk yaptığımız işlem zamanlayıcıyı iptal etmek olacaktır. Daha sonra her nesnenin x ve y özelliklerine başlangıç değerlerini (orijinal koordinat değerlerini: baseX, baseY) atamaktır.

Son olarak yeni animasyon değerlerinin hesaplanması için `animasyonHesapla()` fonksiyonu ve nesnelerin orijinal noktalarına çizdirilmesi için `draw()` fonksiyonu çağrılmıştır.

```
var mouseOut = function () {
            clearTimeout(timer1);
            for (var i = 0; i < daireArray.length; i++) {
                var tempDaire = daireArray[i];
                tempDaire.x = tempDaire.baseX;
                tempDaire.y = tempDaire.baseY;
            }
            animasyonHesapla();
            draw();
        }
```

Tarayıcı Desteği

Yukarıda anlatılan metotlar, özellikler için tarayıcı desteğinin ne durumda olduğu aşağıda listelenmiştir. Tablolarda tarayıcı **Görüntüleme Motoru** (*Layout Engine*) adı yerine tarayıcı sürüm adı kullanılmıştır. Özellik ya da metotlar tarayıcının belirtilen sürümü ve sonra çıkan sürümleri tarafından desteklenir.

Canvas Elemanı Metodları

Metodlar	Internet Explorer	Firefox	Opera	Safari
`getContext()`	IE9+	Firefox 1.5+	Opera 9.0	Desteği var
`Context Type "2d"`	IE9+	Firefox 3.5+	Opera 10.6	Desteği var
`Context Type "WebGL"`	Yok	Kısmen	Yok	Kısmen
`toDataURL()`	IE9+	Firefox 1.5+	Opera 9.5	Desteği var

CanvasRenderingContext 2D (context2d) Özellik ve Metodları

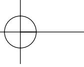
Metodlar / Özellikler	Internet Explorer	Firefox	Opera	Safari
`strokeStyle`	IE9+	Firefox 1.5+	Opera 9.0	Desteği var
`fillStyle`	IE9+	Firefox 1.5+	Opera 9.0	Desteği var
`globalAlpha`	IE9+	Firefox 1.5+	Opera 9.0	Desteği var

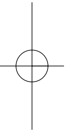
globalCompositeOperation	IE9+	Firefox 1.5+	Opera 9.0	Desteği var
shadowColor	IE9+	Firefox 3.5+	Opera 10.5+	Desteği var
shadowOffsetX	IE9+	Firefox 3.5+	Opera 10.5+	Desteği var
shadowOffsetY	IE9+	Firefox 3.5+	Opera 10.5+	Desteği var
shadowBlur	IE9+	Firefox 3.5+	Opera 10.5+	Desteği var
lineWidth	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
lineCap	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
lineJoin	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
miterLimit	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
font	IE9+	Firefox 3.5+	Desteği var	Desteği var
textAlign	IE9+	Firefox 3.5+	Opera 10.6+	Desteği var
textBaseline	IE9+	Firefox 3.5+	Opera 10.6+	Desteği var
save()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
restore()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
fillRect()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
strokeRect()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
clearRect()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
beginPath()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
moveTo()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
lineTo()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
quadraticCurveTo()	IE9+	Firefox 2.0+	Opera 9.0+	Desteği var
bezierCurveTo()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
closePath()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
arc()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
rect()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
fill()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
stroke()	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var

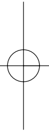
`clip()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`createLinearGradient()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`createRadialGradient()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`createPattern()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`scale()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`rotate()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`translate()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`transform()`	IE9+	Firefox 1.5+	Opera 10.6+	Desteği var
`fillText()`	IE9+	Firefox 3.5+	Opera 10.6+	Desteği var
`strokeText()`	IE9+	Firefox 3.5+	Opera 10.6+	Desteği var
`measureText()`	IE9+	Firefox 3.5+	Opera 10.6+	Desteği var
`drawImage()`	IE9+	Firefox 1.5+	Opera 9.0+	Desteği var
`createImageData()`	IE9+	Firefox 3.5+	Opera 11+	Desteği var
`getImageData()`	IE9+	Firefox 3+	Opera 10.6+	Desteği var
`putImageData()`	IE9+	Firefox 4.0	Opera 10.6+	Desteği var

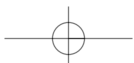


Internet Explorer 7 ve 8 sürümlerinde canvas özellik ve metotlarına kısmen destek sağlamak için **Explorercanvas** (*excanvas.js*) JavaScript kütüphanesini kullanabilirsiniz.









HTML5 VE SVG

7

SVG (**S**_calable_ **V**_ector_ **G**_raphics_), XML tabanlı 2 boyutlu çizimler oluşturmak için kullanılan bir işaretleme dilidir. SVG,vektörel tabanlı grafikler oluşturmak için kullanılır ve bir W3C bildirimi olarak diğer W3C bildirimleri ile (Örneğin; XLS, DOM) birlikte çalışabilir. SVG dilini JavaScript ile beraber kullanılıp interaktif ve dinamik çizimler yapabilirsiniz.

Hatırlarsanız canvas bölümünde çizim işlemleri programatik olarak JavaScript yardımıyla yapılmaktaydı. Alternatif olarak SVG etiketlerini kullanarak çizim işlemleri yapabilirsiniz. W3C tarafından desteklenen bu teknolojinin **SVG 1.1** (*Second Edition*) halihazırda tarayıcılar tarafından büyük oranda desteklenmekle beraber W3C yazarları SVG 2.0 sürümünü geliştirmeye devam etmektedirler.

Ayrıca görsel olarak SVG dosyaları (grafikleri) oluşturmak için kullanabileceğiniz programlar (örneğin; Adobe Illustrator CS5) ve uygulamalar mevcuttur.

SVG VE HTML5 KULLANIMI

SVG içeriklerini ayrı bir dosyada saklayabilir ve tarayıcılar arasında farklılık göstermekle beraber `<embed>`, `<object>` ya da `<iframe>` etiketlerini kullanarak web belgesi içerisine ekleyebilirsiniz. Ancak HTML5 yapısı ile beraber tarayıcıların destekledikleri en önemli özellik inline (*satır içi*) SVG kodları yazabilmemizdir. Yani HTML5 yapısı içerisinde SVG etiketlerini diğer etiketler gibi direk sayfa içerisinde kullanabilirsiniz.

Aşağıda inline SVG formatı görülmektedir.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>SVG</title>
    <style type="text/css">
    </style>
    <script type="text/javascript">
    </script>
</head>
<body>
    <svg width="" height="" xmlns="http://www.w3.org/2000/svg">
        <!--width,height özellikleri svg çizim alanının genişliği ve yüksekliğini tanımlar.
        .
        SVG Etiketleri
        .
        -->
    </svg>
</body>
</html>
```

"Çizim işlemleri için canvas ya da SVG hangisini kullanmaıyım?" şeklinde bir soru aklınıza gelebilir. Şöyle kısaca özetleyelim: Yüksek ve sürekli performans için canvas elemanı tercih etmelisiniz. Ayrıca canvas elemanının piksel tabanlı görüntü işleme olanağı sağlaması en büyük avantajlarından bir tanesidir. Diğer bir yandan, SVG teknolojisinin canvas elemanına göre en büyük artısı tüm çizim işlemlerinin etiketlerle yapılması ve her çizim elemanına DOM yöntemleri ile kolayca ulaşabilmemizdir. Bununla beraber SVG yapısının çözünürlükten bağımsız çizimler oluşturması diğer bir artısıdır. Canvas elemanını oyun programlama, interaktif ve piksel tabanlı görüntü düzenleme, görüntü analizi gibi uygulamalarda diğer yandan SVG yapısını çözünürlükten bağımsız kullanıcı arayüzleri oluştururken tercih edebilirsiniz.

SVG yapısı içerisindeki bir etiketin özelliğine (*attribute*) değer atamak ya da değiştirmek için `setAttribute()`, özelliğin aldığı değeri öğrenmek için `getAttribute()` metotlarını kullanabilirsiniz. Bu metotlar **DOM Level 1 Core** ile tanımlanmıştır.



Aşağıdaki örneklerde sadece HTML5 **body** elemanı içerisine inline olarak eklenecek **<svg>** elemanının içeriği verilecektir. Bu kodların **<body>** elemanı içerisine yazıldığını unutmayınız!

Dikdörtgen Çizimi <rect>

Örnek:

```
<svg xmlns="http://www.w3.org/2000/svg" height="100" width="300">
    <rect fill="blue"
       stroke="gray"
       stroke-width="5"
       width="150"
       height="70"
       x="10"
       y="10"
       rx="20"
       ry="30"/>
       <!--
       width:Dikdörtgen genişliği
       height:Dikdörtgen yüksekliği
       >>
       SVG elemanı ile tanımlanan koordinat sisteminde dikdörtgenin sol üst köşesinin
       koordinatları x,y özellikleri ile ayarlanır.
       >>
       Yuvarlak köşeli dikdörtgenler oluşturmak için rx,ry özellikleri kullanılır.
       SVG dikdörtgen köşelerini yuvarlatmak için rx,ry özellikleri ile tanımlı bir elips oluşturur.
       Elipsin x eksenindeki yarıçapı rx, y eksenindeki yarıçapı ry özellikleri ile ayarlanır.
       >>
       fill, stroke, stroke-width ... özellikleri presentation attributes (stil/sunum özellikleri)
       olarak adlandırılır. İsterseniz bu özelliklere yukarıdaki gibi ya da style özelliği
       içerisinde değer atayabilirsiniz.
       style="fill:blue;stroke:gray;stroke-width:5" gibi.
       -->
    </svg>
```

Sonuç:

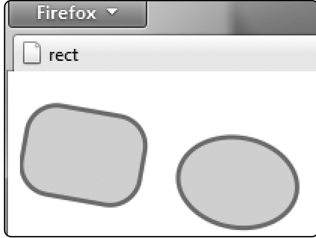


Ekran görüntüsü Firefox 4 ile elde edilmiştir.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>rect</title>
    <script type="text/javascript">
        var init = function () {
            var gElement = document.getElementsByTagName("g")[0];
            gElement.setAttribute("style",
"fill:#FCD526;stroke:gray;stroke-width:3");
        }
    </script>
</head>
<body onload="init();">
<svg xmlns="http://www.w3.org/2000/svg" height="100" width="300">
    <g id="g1" transform="rotate(10)">
    <rect x="10" y="10" width="80" height="60" rx="20" ry="20"/>
    <rect x="115" y="10" width="80" height="60" rx="40" ry="40"/>
    </g>
       <!--
       <g> etiketi,grafik çizimi yapan elemanları gruplandırmak için kullanılır.
       Yukarıdaki örnekte "g1" id tanımlamasına sahip konteynır, g elemanına dinamik
       olarak stil özellikleri tanımlanacaktır. Ayrıca g elemanı içerisinde bulunan elemanlara
       transform özelliği ile 10 derecelik bir döndürme uygulanmıştır. Burada döndürme
       açısı derece cinsinden tanımlanır.
       -->
    </svg>
</body>
</html>
```

Ekran görüntüsü:



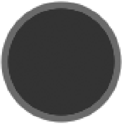
Firefox 4 ekran görüntüsü.

Daire Çizimi <circle>

Örnek:

```
<svg xmlns="http://www.w3.org/2000/svg" height="100" width="300">
<circle fill="blue"
    stroke="gray"
    stroke-width="5"
    cx="100"
    cy="50"
    r="40"/>
    <!--
    SVG elemanı ile tanımlanan koordinat sisteminde dairenin merkez noktası cx,cy özellikleri
    ile tanımlanır. r özelliği ile daire yarıçapı ayarlanır.
    -->
</svg>
```

Sonuç:



Ekran görüntüsü Firefox 4 ile elde edilmiştir.

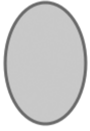
Elips Çizimi <ellipse>

Örnek:

```
<svg xmlns="http://www.w3.org/2000/svg" height="200" width="300">
     <ellipse fill="rgb(247,208,34)" stroke="gray" stroke-width="3"
    cx="100"
    cy="100"
    rx="40"
```

```
    ry="60"/>
        <!--
        SVG elemanı ile tanımlanan koordinat sisteminde elipsin merkez noktası cx,cy
        özellikleri ile tanımlanır.
        >>
        Elipsin yatay eksendeki yarıçapı rx, dikey eksendeki yarıçapı ry özellikleri ile ayarlanır.
        -->
</svg>
```

Sonuç:



Ekran görüntüsü Firefox 4 ile elde edilmiştir.

Çizgi Çizimi <line>

Örnek:

```
<svg xmlns="http://www.w3.org/2000/svg" height="200" width="300">
    <g style="stroke:red;stroke-width:3">
    <line x1="10"  y1="10" x2="100"   y2="10" />
    <line x1="100" y1="10" x2="100"   y2="80" />
    <line x1="100" y1="80" x2="10"    y2="10" />
    </g>
        <!--
        SVG elemanı ile tanımlanan koordinat sisteminde x1,y1 özellikleri ile çizgi başlangıç
        noktası x2,y2 özellikleri ile çizgi bitiş noktası tanımlanır.
        Unutmayın! Etiketler içindeki belirtilen özelliklere JavaScript yardımıyla ulaşıp
        değerlerini öğrenebilir ya da değiştirebiliriz.
        -->
</svg>
```

Sonuç:



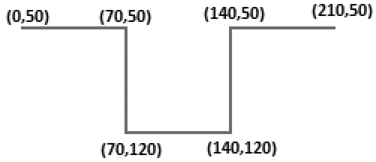
Ekran görüntüsü Firefox 4 ile elde edilmiştir.

Çoklu Çizgi Çizimi <polyline>

Örnek:

```
<svg xmlns="http://www.w3.org/2000/svg" height="200" width="300">
    <polyline style="fill:white;stroke:red;stroke-width:2"
    points="0,50 70,50 70,120 140,120 140,50 210,50"/>
       <!--
       svg elemanı ile tanımlanan koordinat sisteminde svg(0,0) points özelliği içerisine
       çizim işleminde kullanılacak x,y koordinatları atanır. points="x1,y1 x2,y2 ...."
       şeklinde kullanılır.
       -->
</svg>
```

Sonuca işaretlemeler yaparak bakalım:



Ekran görüntüsü Firefox 4 ile elde edilmiştir.

Çokgen Çizimi <polygon>

Örnek:

```
<svg xmlns="http://www.w3.org/2000/svg" height="200" width="300">
     <polygon fill="gray" stroke="lightblue" stroke-width="10"
           points="110,50 160,70 166,122 136,163 82,163 50,122 110,110" />
       <!--
       points özelliği içerisine çizim işleminde kullanılacak x,y koordinatları atanır.
       points="x1,y1 x2,y2 x3,y3 ..." şeklinde kullanılır. <polyline> elemanından farklı
       olarak çizim yolunu kapatır(closePath() metodu gibi)
       -->
</svg>
```

Ekran görüntüsü:



Ekran görüntüsü Firefox 4 ile elde edilmiştir.

Yol Çizimi <path>

İçerisinde düz ve eğimli çizgileri barındıran yol çizimleri oluşturmak için kullanılır.

Örnek:

```
<svg xmlns="http://www.w3.org/2000/svg" height="200" width="300">
<path d="M 20 120 L 90 10  150 120 C 100 50 50 100 20 120 "
fill="red" stroke="blue" stroke-width="3" />
</svg>
```

Ekran görüntüsü:



Ekran görüntüsü Firefox 4 ile elde edilmiştir.

Yol bilgisini içeren komut satırı `d` özelliğine atanır. Yol bilgisi komutlarla oluşturulur ve her bir komut kendisinden sonra yazılan koordinat bilgilerine göre yol parçasını ya da parçalarını çizer (`M`, `Z` komutları yol çizmez tanımlama yapar). Komut isimleri yerine aşağıda tanımlanan karakterler kullanılır. `Z(closePath)` komutu; yol çizimini kapatacağından kendinden sonra bir koordinat bilgisi almaz. `M(moveTo)` komutundan sonra yazılan, koordinatlar çizime başlangıç noktasını tanımlar. `d` özelliğine atanan veri okunabilirliğin artması için birden fazla satıra bölünebilir.

Aşağıdaki komutlar büyük harfle yazılırlarsa **mutlak pozisyonlandırma** kullanılır. Yani yol boyunca orijin noktası daima svg elemanının sol üst köşesi olarak kabul edilir. Fakat küçük harfle yazılırlarsa orijin noktası olarak kendilerinden önce yol çiziminde kullanılan en son noktayı kullanırlar. Komut satırında ilk kullanılan `moveTo` komutu için `M` karakterinin küçük yazılması bir şey değiştirmez. Çünkü henüz yol çiziminde kullanılan bir nokta bilgisi yoktur. Ya da `Z`, `z` komutlarının birbirinden farkı yoktur; ikisi de yol çizimini kapatır.

Komutlar:

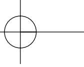
- **M (moveto):** `M (X,Y)+` şeklinde kullanılır.
- **L (lineto):** `L (X,Y)+` şeklinde kullanılır. Burada `L(x,y)+` formatıyla anlatılmak istenen; siz `L` karakterinden sonra birden fazla `x,y` çifti yazabilirsiniz. Yani `d="M 0 0 L 50 50 100 50"` gibi. Bu durumda `(0,0)` noktasından `(50,50)` noktasına bir çizgi ve yine `(50,50)` noktasından `(100,50)` noktasına bir çizgi çizilir.

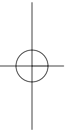
- **`H` (horizontal lineto-yatay çizgi):** `H(x)+` şeklinde kullanılır.
- **`V` (vertical lineto-dikey çizgi):** `V(y)+` şeklinde kullanılır.
- **`C` (curveto):** `(x1 y1 x2 y2 x y)+` şeklinde kullanılır. `(x1,y1)` ve `(x2,y2)` kontrol noktalarıdır. `(x,y)` değer çifti çizim bitiş noktasını tanımlar. Bu tanım diğer komutlar içinde geçerlidir.
- **`S` (smooth curveto):** `(x2 y2 x y)+` şeklinde kullanılır.
- **`Q` (quadratic Belzier curve):** `(x1 y1 x y)+` şeklinde kullanılır.
- **`T` (smooth quadratic Belzier curveto):** `(x y)+` şeklinde kullanılır.
- **`A` (elliptical Arc):** `(rx ry x-axis-rotation large-arc-flag sweep-flag x y)+` şeklinde kullanılır.
- **`Z` (closePath):** `"z"` ya da `"Z"` şeklinde kullanılır.

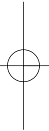
Yukarıda SVG ile temel çizimlerin nasıl yapıldığı anlatılmıştır. Aslında SVG'nin tamamı ayrı bir kitabın konusudur. Bu kitap için bu kadar SVG bilgisinin yeterli olduğu kanısındayım. Çünkü SVG grafiklerini elle kod yazarak oluşturmak yerine var olan görsel editörleri kullanabilir ve gerektiği yerlerde bu kodlara elle müdahale edebilirsiniz. Ayrıca SVG ile yapabileceğiniz ve bizim bu bölümde yer vermediğimiz birçok işlem mevcuttur. Bunlar; **Gradients** ve **Patterns** dolguları, SVG elemanı içinde link alanları oluşturma **Clipping**, **Masking** ve **Compositing** işlemleridir.

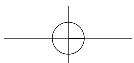


Inline-SVG şu an için **IE 9.0**, **Firefox 4.0** tarafından desteklenmektedir.









HTML5 AUDIO ve VIDEO ELEMANLARI

8

HTML5, herhangi bir plug-in kullanmadan tarayıcıların multimedya (ses, video) dosyalarını çalıştırılabilmesini sağlar. Ayrıca kendi medya denetleyicilerinizi (oynat, durdur gibi) oluşturup ses ya da videoyu JavaScript yardımıyla kontrol edebilir ve CSS ile bu elemanlara görsel özellikler ekleyebilirsiniz. Web sayfasına eklediğiniz medya içeriğinin JavaScript yardımıyla kontrol edilebilir olması önemlidir. Bu durum bize sayfadaki diğer elemanların herhangi bir olayına ya da kullanıcının diğer elemanlarla girdiği herhangi bir etkileşime bağlı olarak ses ya da video dosyaları ile ilgili işlem yapma imkanı sunar. HTML5, medya dosyalarını web sayfaları içerisine gömmek için `<audio>` ve `<video>` elemanlarını tanımlar. Şimdi bu elemanlara ayrıntılı bir şekilde bakalım.

<AUDIO>

Web sayfaları içerisine ses dosyaları eklemek için kullanılan elemandır. Aşağıdaki tabloda tarayıcıların hangi formattaki ses dosyalarına destek verdikleri listelenmektedir.

Dosya Türü	Internet Explorer	Firefox	Opera	Safari
Mp3 (*.mp3*)	9.0	–	–	Desteği var (QuickTime ile)
Ogg Vorbis (*.ogg*)	–	3.5+	10.5+	–
Wav (*.wav*)				Desteği var (QuickTime ile)

Özellikleri: [Standart Özellikler], src, controls, autoplay, preload, loop

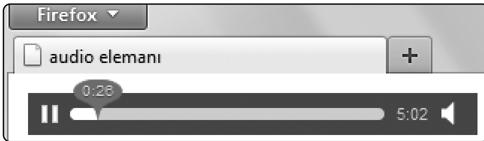
SRC

audio elemanı tarafından çalınacak kaynak ses dosyasını tanımlamak için kullanılır. Bu özelliğe ses dosyasının saklandığı bir URL adresi de yazılabilir.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>audio elemanı</title>
    <style type="text/css">
        audio
        {
            margin:5px;
        }
    </style>
</head>
<body>
    <audio src="audio1.ogg" controls>
    </audio>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü.

CONTROLS

Denetim çubuğunun (kontrol düğmelerinin) gösterilip gösterilmeyeceğini ayarlamak için kullanılır. Kontrol düğmelerinin gösterilmemesini sağlayıp, kendi oluşturduğunuz kontrol düğmelerini kullanabilir ya da autoplay özelliğini kullanıp, ses dosyasının sayfa yüklendiğinde otomatik olarak çalıştırılmasını sağlayabilirsiniz. Bu özelliği tek başına değer atamadan kullanabilirsiniz.

```
<audio src="ses.ogg" controls>
</audio>
```

Aşağıda tarayıcılar tarafından varsayılan denetim çubukları görünmektedir.



Firefox 4



Opera 11



IE 9

AUTOPLAY

Sayfa yüklendiğinde, ses dosyası verisi kullanıma hazır olduğunda tarayıcının ses dosyasını otomatik olarak çalıştırmasını sağlar. Bu özelliği tek başına değer atamadan kullanabilirsiniz.

```
<audio src="ses.ogg" autoplay>
</audio>
```

PRELOAD

Sayfa yüklendiğinde medya içeriğinin çalışmaya hazır olması için ön belleğe alınmasını sağlar. Eğer `autoplay` özelliği kullanılmışsa, bu işlem zaten otomatik olarak yapılır. `none`, `metadata`, `auto` değerlerinden birini alabilir.

`none` değeri atanırsa medya içeriği için preload işlemi yapılmaz. **`metadata`** değeri atanırsa performans açısından önemli olan medya içeriği ile ilgili çeşitli bilgiler (boyutu, toplam süre, kaynak listesi) için ön yükleme yapılır. **`auto`** değeri atanırsa ön yükleme işlemi otomatik olarak tüm medya içeriği için yapılır.

LOOP

Ses dosyası sonlandığında tekrar baştan çalıştırılmasını sağlamak için kullanılır. Bu özelliği tek başına değer atamadan kullanabilirsiniz.

```
<audio src="ses.ogg" loop>
</audio>
```

Programatik olarak bir `audio` elemanını oluşturup kullanmanız da mümkündür. Aşağıdaki örneği inceleyiniz.

Örnek:

```
<script type="text/javascript">
       /*start() sayfa yüklendiğinde çalışacak fonksiyon*/
       var start = function () {
            var audio = new Audio("audio1.ogg");
```

```
            audio.controls = true;
            document.body.appendChild(audio);
            /*Ya da aşağıdaki gibi bir format kullanmanızda mümkündür.
            var audio = new Audio();
            audio.setAttribute("src", "audio1.ogg");
            audio.controls = true;
            document.body.appendChild(audio);
            */
        }
    </script>
```

<VIDEO>

Web sayfaları içerisine video dosyaları eklemek için kullanılan elemandır. Herhangi bir eklenti kullanmadan video içeriklerinin tarayıcılar tarafından oynatılmasını sağlayan bir standart sunar. Aşağıdaki tabloda tarayıcıların hangi formattaki ses dosyalarına destek verdikleri görünmektedir.

Dosya Türü	Internet Explorer	Firefox	Opera	Safari
Ogg	–	3.5+	10.5+	–
WebM	–	4.0	10.6+	–
MPEG 4	9.0	–	–	Desteği var.

Özellikleri: `[Standart Özellikler]`, `src`, `controls`, `autoplay`, `preload`, `loop`, `poster`, `width`, `height`

`src`, `controls`, `autoplay`, `preload`, `loop` özellikleri <audio> elemanının belirtilen özellikleri ile aynıdır. width, height özellikleri video elemanının genişliğini ve yüksekliğini ayarlamak için kullanılır.

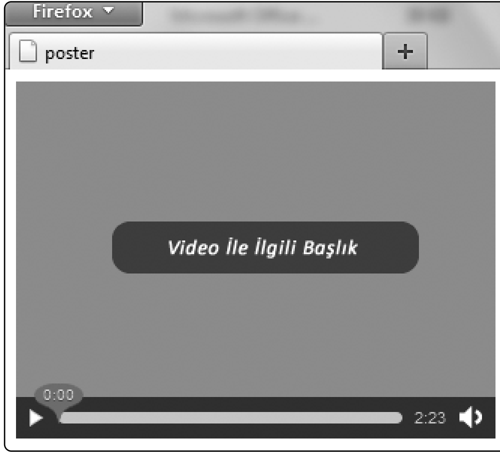
POSTER

Video içeriğini temsil eden bir resim tanımlaması yapmak için kullanılır. Bu resim, medya içeriği çalıştırılıncaya kadar video elemanı içinde gösterilir.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
```

```
    <title>poster</title>
</head>
<body>
    <video id="vd" src="cartoon.ogg" controls poster="pre.png">
    </video>
</body>
</html>
```



Firefox 4 ekran görüntüsü.

<SOURCE>

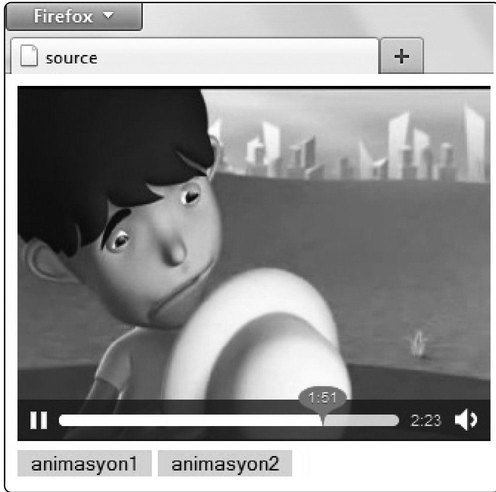
<audio> ve <video> elemanlarına alternatif, birden fazla kaynak dosya tanımlamak için kullanılır. Tarayıcı, medya elemanları (audio, video) içerisinde source elemanları ile yapılan tanımlama sırasına göre kendisi tarafından desteklenen ilk medya içeriğini oynatır. source elemanlarıyla kaynak içerikler tanımlayarak, audio ve video elemanlarının çalıştıracakları medya içeriklerini dinamik olarak değiştirebilirsiniz.

Özellikleri: `[Standart Özellikler],src, type, media`

src özelliği ile kaynak medya dosyası tanımlanır. type özelliği ile medya kaynağı için **MIME type** (dosyaları sınıflandırmak için kullanılan içerik türü) tanımlaması yapılır. Medya elemanının çalıştırılacağı hedef ortama göre çeşitli tanımlamalar (width, height, resolution gibi) yapmak için media özelliği kullanılır.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>source</title>
    <style type="text/css">
    input:nth-of-type(n)
    {
        background-color:lightblue;
        border:none;
    }
    </style>
    <script type="text/javascript">
        var start = function () {
            document.getElementById("vd").src =
document.getElementsByTagName("source")[0].src;
        }
        var start_ = function () {
            document.getElementById("vd").src =
document.getElementsByTagName("source")[1].src;
        }
    </script>
</head>
<body onload="start();">
    <video id="vd" controls>
    <source src="cart1.ogg"/>
    <source src="cart2.ogg"/>
    </video>
    <br />
    <input type="button" value="animasyon1" onclick="start();" />
    <input type="button" value="animasyon2" onclick="start_();" />
</body>
</html>
```



Firefox 4 ekran görüntüsü.

Yukarıdaki örnekte sayfa yüklendiğinde çalışacak `start()` fonksiyonu ile `video` elemanının `src` özelliğine ilk `source` elemanı ile tanımlanan medya kaynağı atanmıştır. Bu durumda sayfa yüklendikten sonra kullanıcı oynat düğmesine tıkladığında video elemanı `cart1.ogg` dosyasını oynatacaktır. Eğer kullanıcı `animasyon2` isimli butona tıklarsa, bu durumda video elemanının oynatacağı kaynak dosya `cart2.ogg` olacaktır.

NOT Yukarıda elemanlar için anlatılan özellikler (*attribute*), programatik olarak elemanların referansları içinde birer özellik (*property*) olarak tanımlıdırlar. Yani video elemanı (DOM yapısı HTML elemanlarını birer nesne olarak tanımlar) için tanımlı olan src özelliği etiket içerisinde tanımlanırsa, attribute programatik olarak elemanının referansı ile kullanılırsa property olarak isimlendirilir.

```
<video src ="kaynak" ></video>            // Attribute
HTMLVideoElement.src [ = "kaynak" ]       // Property
```

Tüm HTML elemanları **HTMLElement** arayüzünden türetilmiştir. Yani HTML elemanları **HTMLElement** arayüzü ile tanımlanan özellik ve metotlara sahiptir. audio elemanı HTMLAudioElement, video elemanı **HTMLVideoElement** DOM arayüzlerini kullanır. Aynı zamanda bu iki media elemanı ortak olarak **HTMLMediaElement** arayüzü ile tanımlanan özellik ve metotları miras alır. HTMLMediaElement arayüzü ile ve elemanların kendi arayüzleri ile tanımlanan özelliklerin bir kısmını (*attribute*) olarak yukarıda anlatmıştık. Şimdi HTMLMediaElement arayüzü ile elemanlar (*nesneler*) için tanımlanan ortak özellik (*property*) ve metotlara bakalım.

HTMLMediaElement Arayüzü ile Tanımlanan Özellikler ve Metotlar

Özellikler (Properties)

Adı	Açıklama
`autoplay, controls, preload, loop`	Daha önce ayrıntılı bir şekilde anlattığımız bu özellikler boolean türünden özelliklerdir. Yani bu özelliklere `true` ya da `false` değerlerinden biri atanabilir. **Kullanımı:** `mediaElemanRef.ozellik[=true││false]`
`src`	`DOMString` türünden olan bu özellik medya kaynağı tanımlaması için kullanılır. **Kullanımı:** `mediaElemanRef.src[=kaynak]` **Örnek:** `<!DOCTYPE html>` `<html>` `<head>` `    <meta charset="utf-8" />` `    <title>src</title>` `    </style>` `    <script type="text/javascript">` `        var start = function () {` `document.getElementsByTagName("audio")[0].src="audio1.ogg";` `        }` `    </script>` `</head>` `<body onload="start();">` `<audio controls></audio>` `</body>` `</html>`
`currentSrc`	Sadece okunabilir olan bu özellik medya kaynağı bilgisini elde etmek için kullanılır. **Kullanımı:** `mediaElemanRef.currentSrc`

	Örnek: `console.log(document.getElementsByTagName("audio")[0].currentSrc);` Yukarıdaki kodu `start()` fonksiyonu içerisine yerleştirip sayfayı çalıştırdığınızda; *"file:///C:/Users/sena/Desktop/medya/audio1.ogg"* şeklinde bir sonuç göreceksiniz.
buffered	Arabelleğe alınmış medya içeriği ile ilgili zaman aralıklarını saklayan `TimeRanges` (sadece okunabilir) türünden bir nesne döndürür. **Kullanımı:** `mediaElemanRef.buffered`
currentTime	Geçerli oynatma süresini saniye olarak döndürür. Daha açık olarak oynatma akışında gelinen noktanın saniye cinsinden değerini verir diyebiliriz. **Kullanımı:** `mediaElemanRef.currentTime [ = value ]`
defaultPlaybackRate, playbackRate	Varsayılan oynatma hızını defaultPlaybackRate özelliği ile ayarlayabilirsiniz. Ya da çalışma anında oynatma hızını değiştirmek için `playbackRate` özelliğini kullanabilirsiniz. `1.0` değeri varsayılan hızı temsil eder. Yukarıdaki özelliklere atanan değer `1.0`'dan küçük ise medya içeriği yavaş, büyük ise daha hızlı oynatılır.
duration	Medya içeriğinin saniye cinsinden uzunluğunu öğrenmek için kullanılır. Sadece okunabilir bir özelliktir. Medya içeriği kullanılabilir durumda, fakat uzunluk bilgisi bilinmiyorsa `NAN`, medya uzunluğu ön tanımlı değil ise `Infinity` değerini alır. **Kullanımı:** `mediaElemanRef.duration`
ended	Oynatma işlemi ile ilgili bilgi elde etmek için kullanılır. Sadece okunabilir bir özelliktir. Oynatma işlemi tamamlanmış ise `true`, tamamlanmamış ise `false` değerini döndürür. **Kullanımı:** `mediaElemanRef.ended`
error	Oynatma akışında meydana gelen en son hatayı saklayan bir `MediaError` (sadece okunabilir) nesnesi döndürür. Eğer herhangi bir hata oluşmamış ise null değerini saklar. `MediaError` nesnesinin `code` özelliğini kullanarak hata nedenini elde edebiliriz. `mediaElemanRef.error.code` `code` özelliği aşağıdaki ön tanımlı değerlerden birini döndürür. `MEDIA_ERR_ABORTED` (sayısal değeri 1) `MEDIA_ERR_NETWORK` (sayısal değeri 2) `MEDIA_ERR_DECODE` (sayısal değeri 3) `EDIA_ERR_SRC_NOT_SUPPORTED` (sayısal değeri 4)

networkState	Ağ üzerinden medya içeriğinin indirilme durumunu elde etmek için kullanılır (Sadece okunabilir bir özelliktir). Bu özelliğin aldığı değerler tarayıcılar arasında farklılık göstermektedir. Özelliğin aldığı ön tanımlı değerler ya da sayısal değerler aşağıda verilmiştir. **Firefox 4:** `EMPTY (0)` Medya içeriği yok. `LOADING(1)` Medya içeriği yükleniyor. `LOADED_METADATA(2)` Medya içeriğinin metadata verisi elde edildi. `LOADED_FIRST_FRAME(3)` Medya içeriğinin ilk karesi elde edildi. `LOADED(4)` Medya içeriğinin tamamı elde edildi. **IE ve HTML5 bildirimi:** `NETWORK_EMPTY (0)` `NETWORK_IDLE (1)` `NETWORK_LOADING (2)` `NETWORK_NO_SOURCE (3)`
readyState	Medya verisinin yüklenmesi (kullanıma hazır olması) ile ilgili daha ayrıntılı bilgiler elde etmek için kullanılır (sadece okunabilir bir özelliktir). Özelliğin aldığı ön tanımlı değerler ya da sayısal değerler aşağıda verilmiştir. `HAVE_NOTHING(0)` Medya içeriği hazır değil. `HAVE_METADATA(1)` Medya içeriğinin metadata verisi hazır. `HAVE_CURRENT_DATA(2)` Geçerli çalma pozisyonu için veri elde edildi. `HAVE_FUTURE_DATA(3)` Geçerli çalma pozisyonuna göre bir sonraki çalma pozisyonu/pozisyonları için veri elde edildi. `HAVE_ENOUGH_DATA(4)` Yeterli bilgi mevcut değil, fakat indirme hızı yüksek medya içeriği kesintisiz olarak sonuna kadar oynatılabilir.
paused	Oynatma işlemi duraklatılmış ise bu özellik `true`, değilse `false` değerini döndürür. Sadece okunabilir bir özelliktir. **Kullanımı:** `mediaElemanRef.paused`

played	Oynatılmış medya içeriği ile ilgili zaman aralıklarını saklayan `TimeRanges` (sadece okunabilir) türünden bir nesne döndürür. **Kullanımı**: `mediaElemanRef.played`
muted	Medya içeriği sessiz olarak ayarlanmış ise `true`, değilse `false` değeri döndürür. **Kullanımı:** `mediaElemanRef.muted`
Diğer Özellikler	`seekable`, `seeking`, `startTime`, `volume`

Metotlar

`HTMLMediaElement` arayüzü ile tanımlanan metotları kullanarak medya içeriğini kontrol edebilirsiniz. Şimdi bu metotları inceleyelim.

canPlayType()

Tarayıcının belirtilen medya formatını destekleme durumunu öğrenmek için kullanılır.

Kullanımı: `mediaElemanRef.canPlayType("medyaTürü")`

Bu metodun geriye döndürdüğü değerlere bakalım.

- **`probably:`** Medya türü tarayıcı tarafından oynatılabilir.
- **`maybe:`** Oynatılabilir olup olmadığı bilinmiyor.

Eğer boş bir string verisi döndürürse, tarayıcı belirtilen medya türünü oynatamıyor.

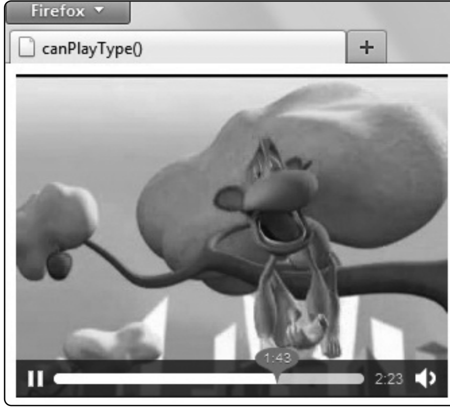
Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>canPlayType()</title>
    <style type="text/css">
        p
        {
            margin-top: 5px;
            width: 170px;
            background-color:crimson;
            text-indent:3px;
            color:White;
```

```
        }
    </style>
    <script type="text/javascript">
        var init = function () {
            var video = document.getElementsByTagName("video")[0];
            if (video.canPlayType("video/mp4") !== "") {
                video.src = "video.mp4";
            } else if (video.canPlayType("video/ogg") !== "") {
                video.src = "video.ogg";
            } else if (video.canPlayType("video/webm") !== "") {
                video.src = "video.webm";
            }
            var p_ = document.querySelector("p");
            var reg = /\w+.\w+$/;
      /*Video elemanının src özelliğini p elemanı içerisine yazdırmak istiyoruz.
      (Video elemanı tarafından oynatılan dosya adını) src özelliğine ulaşmak istediğimizde
      aşağıdaki gibi bir ifade elde ederiz.
      "file:///C:/Users/sena/Desktop/medya/video.uzantı" bu metin içerisinden sadece
      dosya adı ve uzantısı bilgisini elde etmek istediğimden bir düzenli ifade nesnesi
      (RegExp) oluşturduk. Belirtilen düzenli ifade ile eşleşen metnin sonundaki dosya adı
      ve uzantısı bilgisini reg.exec(vSource) komut satırı ile elde edip bir metin düğümü
      olarak p elemanı içerisine ekledik.*/
            vSource = video.src;
            p_.appendChild(document.createTextNode("oynatılan dosya:"
+ " " + reg.exec(vSource)));
        }
    </script>
</head>
<body onload="init();">
    <video controls></video>
    <p>
    </p>
</body>
</html>
```

Yukarıdaki örnekte aynı medya içeriği farklı tarayıcılarda çalışabilmesi için üç farklı formatta tanımlanmıştır. Bu durumda tarayıcı desteklediği formattaki medya içeriğini oynatacaktır.

Ekran görüntüleri:



Firefox 4 ekran görüntüsü.



IE9 ekran görüntüsü.

LOAD()

Medya elemanı tarafından oynatılacak kaynak içeriği sunucudan tekrar yüklemek için kullanılır.

Kullanımı: `mediaElemanRef.load()`

PLAY()

Medya içeriğini oynatmak için kullanılır. Eğer sayfa yüklendikten sonra medya elemanının `src` özelliğini değiştirdiyseniz, `play()` metodundan önce `load()` metodunu kullanmalısınız. Bu metot çalıştırıldığında `paused` özelliği `false` değerini alacaktır.

Kullanımı: `mediaElemanRef.play()`

PAUSE()

Oynatma işlemi duraklatmak için kullanılır. Bu metot çalıştırıldığında `paused` özelliği `true` değerini alacaktır.

Kullanımı: `mediaElemanRef.pause()`

Örnek: Bu örnekte video elemanı için tarayıcının sağladığı kontrol çubuğu yerine kendi oluşturduğunuz kontrol düğmelerini kullanacağız. Şimdilik ilk örneğimiz basit olması açısından **oynat/durdur** ve **ses aç/ses kapat** işlevlerini yerine getirecek iki tane düğmeden oluşacak.

HTML sayfası içeriği:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>play(),pause()</title>
    <style type="text/css">
        video
        {
            border: 1px solid gray;
        }
    </style>
    <script type="text/javascript" src="video.js"></script>
</head>
<body onload="init();">
    <video id="video1" src="bun.ogv" onended="end();" width="400">
    </video><br />
    <img alt="pausePlay" src="play.png" onclick="pausePlay();" />
    <img alt="pmute" src="ses2.png" onclick="mute();" />
</body>
</html>
```

video.js isimli bir JavaScript dosyası oluşturup içeriğini yazmaya başlayalım...

Komut dosyası içerisinde kullanacağımız değişkenleri direk `<script>` etiketi içerisinde tanımlayıp, bu değişkenlere `init()` fonksiyonu içerisinde değerler atayacağız. Amacımız; tüm fonksiyonlardan belirtilen değişkenlere ulaşabilmektir.

```
var videoEl, img1, img2;
var init = function () {
    videoEl = document.getElementsByTagName("video")[0];
    img1 = document.getElementsByTagName("img")[0];
    img2 = document.getElementsByTagName("img")[1];
}
```

oynat/durdur işlevini yerine getirmek için ilk `img` elemanı kullanılacaktır. Sayfa yüklendiğinde video elemanının `paused` özelliği `true` değerini alır. Çünkü video elemanı için oynatma işlemi sayfa yüklendiğinde otomatik olarak başlamamıştır. Aşağıda ilk `img` elemanı için kaynak olarak kullanacağımız resimler görülmektedir.



Sayfa ilk yüklendiğinde ya da video içeriği duraklatıldığında ilk `img` elemanının kaynak tanımlaması `"play.png"` iken, video oynatıldığında kaynak tanımlaması `"pause.png"` şeklinde olacaktır.

Kullanıcı ilk img elemanına tıkladığında `pausePlay()` fonksiyonu çalışacaktır. Bu fonksiyon içerisinde video elemanının `paused` özelliği kontrol edilecek eğer bu özelliğin değeri `true` ise video elemanı oynatılacak (`play()` metodu kullanıldığından `paused` özelliğinin yeni değeri `false` olur) ve ilk resmin kaynak tanımlamasını `"pause.png"` olarak değiştirilecektir. Eğer `paused` özelliğinin değeri `false` ise medya içeriği oynatılıyor demektir. Bu durumda medya içeriği duraklatılacak (`pause()` metodu kullanıldığından `paused` özelliğinin yeni değeri `true` olur) ve ilk resmin kaynak tanımlaması `"play.png"` olarak değiştirilecektir.

```
var pausePlay = function () {
    if (videoEl.paused) {
        videoEl.play();
        img1.src = "pause.png";
    } else {
        videoEl.pause();
        img1.src = "play.png";
    }
}
```

Ses kontrolü için ikinci `img` elemanı kullanılacaktır. İkinci `img` elemanına tıklandığında `mute()` fonksiyonu çalışır. Aşağıda ikinci `img` elemanı için kaynak olarak kullanılacak resimler görünmektedir.


ses1.png


ses2.png

Sayfa yüklendiğinde `muted` özelliği varsayılan olarak `false` değerini alır. Bu fonksiyon içerisinde video elemanının `muted` özelliğini kontrol edeceğiz. Eğer bu özellik `true` ise (ses kapalı) `muted` özelliğine `false` değerini atayıp (sesi açıp) ikinci `img` elemanının kaynak tanımlamasını değiştiriyoruz. Eğer `muted` özelliğinin değeri `false` ise (ses açık) `muted` özelliğine `true` değerini atayıp (sesi kapatıp) ikinci `img` elemanının kaynak tanımlamasını değiştiriyoruz.

```
var mute = function () {
    if (videoEl.muted) {
        videoEl.muted = false;
```

```
          img2.src = "ses2.png";
     } else {
          videoEl.muted = true;
          img2.src = "ses1.png";
     }
}
```

Son olarak oynatma işlemi tamamlandığında çalışacak `end()` fonksiyonu ile `img` elemanlarına varsayılan resimleri ve `muted` özelliğine `false` değerini atıyoruz.

```
var end = function () {
     img1.src = "play.png";
     videoEl.muted = false;
     img2.src = "ses2.png";
}
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü.

Medya İçeriğini Kontrol Etmek İçin Kullanılabilecek Olaylar

Medya elemanlarını JavaScript yardımıyla kontrol etmek için kullanabileceğiniz olaylara bakalım.

Olay Adı	Açıklama
canplay	Bu olay, sayfa yüklendikten sonra medya içeriğinin oynatılması için yeterli tamponlanmış veri elde edildiğinde oluşur. **Ön koşul ile tanımlama:** readyState özelliği, HAVE_FUTURE_DATA ya da HAVE_ENOUGH_DATA değerlerinden birini aldığında canplay olayı oluşur.
canplaythrough	Sayfa yüklendikten sonra tarayıcı medya içeriğini tamponlama işlemi için durmak zorunda kalmadan, sonuna kadar oynatacağını hesaplarsa, bu olay meydana gelir. canplaythrough olayı canplay olayından sonra oluşur. **Ön koşul ile tanımlama:** readyState özelliği, HAVE_ENOUGH_DATA değerini aldığında canplaythrough olayı oluşur.
durationchange	Medya elemanı yüklendikten sonra ya da medya elemanı için kaynak dosya tanımlaması değiştirildiğinde yani duration özelliğinin aldığı değer güncellendiğinde (hesaplandığında) bu olay meydana gelir.
emptied	Medya kaynak verisi elde edilemezse bu olay gerçekleşir. **Ön koşul ile tanımlama:** networkState özelliği NETWORK_EMPTY değerini aldığında emptied olayı oluşur.
ended	Oynatma işlemi tamamlandığında oluşur. Bu olay gerçekleştiğinde ended özelliği true değerini alır.
error	Medya elemanı ile ilgili bir hata meydana geldiğinde bu olay oluşur. error özelliğini kullanarak hata ile ilgili bilgilere ulaşabilirsiniz.
loadeddata	İlk çalma pozisyonu için veri elde edildiğinde bu olay oluşur. Bu olay canplay olayından önce loadedmetadata olayından sonra meydana gelir.
loadedmetadata	Medya içeriği ile ilgili metadata verisi elde edildiğinde oluşur. Metadata verisi medya içeriği ile ilgili çeşitli bilgilerdir (Boyutu, toplam süre, kaynak listesi gibi).
loadstart	Tarayıcı medya içeriğini yüklemeye başladığında bu olay oluşur. **Ön koşul ile tanımlama:** networkState özelliği, NETWORK_LOADING değerini aldığında loadstart olayı oluşur.

pause	Oynatma işlemi duraklatıldığında oluşur.
play	Medya içeriği oynatıldığında/çalıştırıldığında oluşur.
progress	Tarayıcı medya verilerini yüklerken oluşur.
ratechange	Medya içeriğini oynatma hızı değiştiğinde bu olay oluşur.
timeupdate	Oynatma pozisyonu değiştiğinde oluşur (`currentTime` özelliğinin aldığı değer değiştiğinde).
volumechange	Medya içeriği sessiz ya da sesli olarak ayarlandığında veya ses seviyesi değiştirildiğinde bu olay oluşur.
Diğer Olaylar	`seeked`, `playing`, `seeking`, `stalled`, `suspend`, `waiting`



Burada yeri gelmişken JavaScript olay yönlendiricileri hakkında bir hatırlatma yapmak istiyorum. HTML elemanlarını için tanımlı olan olaylara JavaScript kodları yazmak için (yani olay gerçekleştiğinde çalışacak JavaScript kodlarını tanımlamak için) olay yönlendiricileri/dinleyicileri kullanılır. HTML elemanlarına olay dinleyicisi eklemek için aşağıdaki yöntemlerden birini kullanabilirsiniz.

HTML Event Handlers (HTML Olay Yönlendiricileri)

Olay ismine on ön eki ekleyerek olay yönlendiricisi adı elde edilir. Bu isim etiket içerisinde bir özellik (*attribute*) olarak kullanılır ve olay oluştuğunda çalışacak JavaScript kodlarını tanımlar.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Event Handler</title>
    <script type="text/javascript">
        var end = function () {
            // Kodlar
        }
    </script>
</head>
<body>
    <video id="video1" controls src="cart.ogg" onended="end();"
     width="400">
    </video>
</body>
</html>
```

DOM Level 0 Event Handlers

Yukarıdaki yöntemle benzer olarak elde edilen olay yönlendiricisi adı eleman referansı için bir özellik (*property*) olarak kullanılır ve olay oluştuğunda çalışacak JavaScript fonksiyonunu tanımlar.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Event Handler</title>
    <script type="text/javascript">
        var init = function () {
            var videoEl = document.getElementsByTagName("video")[0];
            videoEl.onended = function () {
                // Kodlar
            }
        }
    </script>
</head>
<body onload="init();">
    <video id="video1" controls src="cart.ogg" width="400">
    </video>
</body>
</html>
```

Bir eleman için tanımlanan olay yönlendiriciyi kaldırmak için `null` değerini kullanabilirsiniz.

```
videoEl.onended = null;
```

DOM Level 2,3 Event Listener

DOM Level 2,3 bildirimleri HTML elemanlarına olay dinleyicisi eklemek için `addEventListener()` metodunu tanımlar.

Kullanımı:

```
HTMLElement.addEventListener(type,listener,useCapture[isteğe Bağlı])
```

`type` parametresi; ile olay adı (olay türünü temsil eden) tanımlanır. Olay adı "on" ön eki almaz.

`listener` parametresi; olay oluştuğunda çalışacak fonksiyonu tanımlar.

useCapture parametresi; `true` ya da `false` değerlerinden birini alır. Bu parametre ile olay dinleyicisinin ekleneceği olay evresi tanımlanır.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Event Handler</title>
    <script type="text/javascript">
        var init = function () {
            var videoEl = document.getElementsByTagName("video")[0];
            videoEl.addEventListener("ended", end, false);
            /*
            Aşağıdaki yazım formatınıda kullanmanız mümkündür.
            videoEl.addEventListener("ended",function(){
            // kodlar
            },false);
            */
        }
        var end = function () {
            // kodlar
        }
    </script>
</head>
<body onload="init();">
    <video id="video1" controls src="cart.ogg" width="400">
    </video>
</body>
</html>
```

Bir eleman için tanımlanan olay dinleyicisini kaldırmak istiyorsanız; `removeEventListener()` metodunu kullanmalısınız.



IE 8 ve alt sürümleri `addEventListener()`, `removeEventListener()` metotlarını desteklemez. Bu metotlar yerine sadece IE tarafından desteklenen `attachEvent()`, `detachEvent()` metotlarını kullanabilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
```

```
    <meta charset="utf-8" />
    <title>Event Handler</title>
    <script type="text/javascript">
        var init = function () {
            var videoEl = document.getElementsByTagName("video")[0];
            if (document.addEventListener) {
              // DOM Level 2,3 Olay Dinleyicisi
                videoEl.addEventListener("ended", function () {
                    // Kodlar
                }, false);
            } else {
                // Sadece IE için olay dinleyicisi
                videoEl.attachEvent("onended", function () {
                    // Kodlar
                });
            }
        }

    </script>
</head>
<body onload="init();">
    <video id="video1" controls src="cart.ogg" width="400">
    </video>
</body>
</html>
```

Bu bölümde kısa bir şekilde olay dinleyicilerini gördük. Aslında tarayıcılar arasındaki farklılıkları gözeterek, bir olay dinleyici nesnesi oluşturup kullanırsanız olabilecek yazım hataları ortadan kaldırmış ve fazla kod yazma derdinden kurtulmuş olursunuz.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Farklı Tarayıcılar için olay dinleyicisi nesnesi</title>
    <script type="text/javascript">
        /*Farklı tarayıcılarda nesnelere olay dinleyicisi(olaya karşı
          fonksiyon)tanımlamak için olay_dinleyicisi isimli bir nesne oluşturacağım ve
          bu nesnenin iki tane metodu olacak.*/
```

```
        var olay_dinleyicisi = new Object();
        olay_dinleyicisi.ekle = function (nesne, olay, metod) {
            if (document.addEventListener) {
                nesne.addEventListener(olay, metod, false);
            } else if (document.attachEvent) {
                nesne.attachEvent(“on” + olay, metod);
            } else {
                var olay_ad = “on” + olay;
                nesne.olay = metod;
            }
        }
        olay_dinleyicisi.kaldir = function (nesne, olay, metod) {
            if (document.removeEventListener) {
                nesne.removeEventListener(olay, metod, false);
            } else if (document.detachEvent) {
                nesne.detachEvent(“on” + olay, metod);
            } else {
                nesne.olay = null;
            }
        }
        var init = function () {
            var videoEl1 = document.getElementsByTagName(“video”)[0];
            var videoEl2 = document.getElementsByTagName(“video”)[1];
            olay_dinleyicisi.ekle(videoEl1, “ended”, end1);
            olay_dinleyicisi.ekle(videoEl2, “ended”, end2);
        }
        var end1 = function () {
            console.log(“videoEl1”);
        }
        var end2 = function () {
            console.log(“videoEl2”);
        }
    </script>
</head>
<body onload=”init();”>
    <video id=”video1” controls src=”cart.ogg”>
    </video>
    <video id=”video2” controls src=”bun.ogv” width=”400”>
    </video>
</body>
</html>
```

Örnek: Bu örnekte yukarıda yaptığımız video player uygulamasına yeni özellikler ekleyeceğiz.

HTML5 sayfası içeriği

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Video Player Uygulaması-II</title>
    <link type="text/css" rel="Stylesheet" href="video.css" />
    <script type="text/javascript" src="video.js">
    </script>
</head>
<body onload="init();">
    <div id="videoPlayer">
        <video id="video1" src="bun.ogv" onended="end();" width="500">
Tarayıcı video elemanını desteklemiyor.
        </video>
        <div id="controls">
            <img alt="playPause" src="play.png" onclick="playPause();" />
            <canvas id="progress" width="300" height="20">
            </canvas>
            <span id="time">_:_</span>
            <img alt="sesAcik" src="ses1.png" onclick="mute();" />
        </div>
    </div>
</body>
</html>
```

video.css dosyası içeriği

```
div#videoPlayer
{
    width: 500px;
    position: relative;
    height: 278px;
}
div#controls
{
    position: absolute;
```

```
        top: 238px;
        left: 0px;
        height: 40px;
        background-color:rgba(0,0,0,0.7);
        width: 500px;
        visibility: hidden;
}
#controls img[alt='playPause']
{
        margin-left: 10px;
        margin-top: 6px;
}
canvas#progress
{
        margin-bottom: 5px;
        margin-left: 4px;
        background-color: White;
}
#controls span
{
        position:absolute;
        left:370px;
        top:5px;
        font-family:Calibri;
        font-size:24px;
        color:#00baff;
        }
#controls img[alt='sesAcik']
{
        position:absolute;
        left:440px;
        top:4px;
}
```

video.js dosyası içeriği

```
var canvas, ctx, wrap, video, controls, imgPlay,imgAudio, spanMsg;
var init = function () {
        canvas = document.getElementsByTagName("canvas")[0];
        wrap = document.querySelector("div#videoPlayer");
        video = document.querySelector("video");
```

```
    controls = document.querySelector("div#controls");
    spanMsg = document.querySelector("#controls span:nth-of-type(1)");
    imgPlay = document.getElementsByTagName("img")[0];
    imgAudio = document.getElementsByTagName("img")[1];
    wrap.addEventListener('mouseover', function () {
        controls.style.visibility = "visible";
    }, true);
    wrap.addEventListener('mouseout', function () {
        controls.style.visibility = "hidden";
    }, true);
    video.addEventListener('timeupdate', progress, false);
    canvas.addEventListener('click', curPositon, false);
}

var playPause = function () {
    if (video.paused) {
        video.play();
        imgPlay.src = "pause.png";
    } else {
        video.pause();
        imgPlay.src = "play.png";
    }
}
var progress = function () {
    ctx = canvas.getContext('2d');
    ctx.fillStyle = "#00baff";
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    var widthCanvas = (video.currentTime / video.duration) * (canvas.width);
    if (widthCanvas > 0) {
        ctx.fillRect(0, 0, widthCanvas, canvas.height);
    }
    var dak = Math.floor(video.currentTime / 60);
    var san = Math.floor(video.currentTime) % 60;
    san = (san > 9) ? san : '0' + san;
    spanMsg.innerHTML = dak + ':' + san;
}
var curPositon = function (event) {
    var event = event || window.event;
    video.pause();
    imgPlay.src = "play.png";
    if (event.offsetX) {
```

```
            var xcor = event.offsetX;
        } else {
            var xcor = event.layerX - canvas.offsetLeft;
        }
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        ctx.fillRect(0, 0, xcor, canvas.height);
        video.currentTime = (xcor * video.duration) / canvas.width;
        video.play();
        imgPlay.src = "pause.png";
    }

    var end = function () {
        imgPlay.src = "play.png";
        video.muted = false;
        imgAudio.src = "ses1.png";
    }

    var mute = function () {
        if (video.muted) {
            video.muted = false;
            imgAudio.src = "ses1.png";
        } else {
            video.muted = true;
            imgAudio.src = "ses2.png";
        }
    }
```

Uygulama içerisinde kullandığımız resim dosyaları:



Ekran görüntüsü:



Firefox 4 ekran görüntüsü.

Şimdi adım adım yukarıdaki uygulamayı nasıl gerçekleştirdiğimizi anlatalım.

İlk olarak HTML sayfası içerisinde `div#videoPlayer` isimli bir div elemanı oluşturduk ve bu div elemanı içerisinde `video` ve `div# controls` elemanlarını tanımladık. `div#controls` elemanını kontrol çubuğu olarak kullanacağız. `div#controls` elemanı içerisinde **Oynat/Durdur**, **Ses aç/Ses kapat** görevlerini yerine getirecek iki tane `img` elemanı kullanacağız. Ayrıca ilerleme çubuğu (*progress bar*) olarak bir `canvas` elemanı ve oynatma süresini (dakika cinsinden) kontrol çubuğunda göstermek için bir span elemanı kullanacağız.

Sayfa yüklendiğinde çalışacak `init()` fonksiyonu içerisinde elemanların referansları alınmıştır (Elemanlara nesne olarak ulaşılmıştır).

Kullanıcı, Mouse ile `div#videoPlayer` elemanı üzerine geldiğinde kontrol çubuğu gözükecek, diğer durumlarda gözükmeyecektir. Bunun için aşağıdaki kodlar yazılmıştır.

```
wrap.addEventListener('mouseover', function () {
        controls.style.visibility = "visible";
}, true);
```

```
wrap.addEventListener('mouseout', function () {
        controls.style.visibility = "hidden";
}, true);
```

`init()` fonksiyonu içerisine `video` ve `canvas` elemanları için olay dinleyiciler eklenmiştir. `playPause()`, `mute()`, `end()` fonksiyonlarının görevlerini bir önceki örnekte anlatılmıştık.

`canvas` elemanı ile oluşturduğumuz ilerleme çubuğunu; video oynatım süresini görsel olarak kullanıcıya göstermek için ve kullanıcının video akışını bu ilerleme çubuğunu kullanarak değiştirebilmesi için kullanacağız.

Video elemanının timeupdate olayında `progress()` fonksiyonu çalışacaktır. Bu fonksiyon her çalıştığında `canvas` elemanı üzerini `clearRect()` metodu ile temizler. Aynı zamanda sol üst köşesi `canvas(0,0)` noktası olan yüksekliği `20px` ve genişliği geçerli oynatma süresine (`currentTime`) bağlı olarak hesaplanan bir dikdörtgen çizer. `currentTime` özelliğine bağlı olarak `progress()` fonksiyonunun her çalışmasında çizilecek dikdörtgenin genişliği artar ve bir ilerleme çubuğu elde edilmiş olur. Peki, çizilecek dikdörtgenin genişliğini `currentTime` özelliğine bağlı olarak nasıl elde ediyoruz? Aşağıdaki koda bakalım...

```
var widthCanvas = (video.currentTime / video.duration) * (canvas.width);
```

`currentTime` (geçerli oynatma süresini saniye olarak döndürür) özelliği `0.0` olduğunda `widthCanvas` değişkeni `0`'dır. `currentTime=duration` olduğunda, `widthCanvas` değişkeninin aldığı değer `canvas.width`, yani `300px` olacaktır. Aslında kurduğumuz mantık basit olmakla beraber amacımız; `currentTime` özelliğinin aldığı değere bağlı olarak genişliği hesaplanan bir dikdörtgeni `canvas` elemanı üzerine çizdirmektir.

Ayrıca `progress()` fonksiyonu içerisinde video akışında gelinen zamanı yani geçerli oynatma süresini dakika cinsinden `span` elemanı içerisine yazdırıyoruz. Bu işlemi aşağıdaki kodları kullanarak yapmaktayız.

```
var dak = Math.floor(video.currentTime / 60);
var san = Math.floor(video.currentTime) % 60;
san = (san > 9) ? san : '0' + san;
```

`Math.floor()` metodu ondalıklı sayıyı kendinden küçük ilk tamsayıya yuvarlar.

```
Math.floor(32.9) >> 32
Math.floor(32.2) >> 32
Math.floor(-32.9) >> -33
```

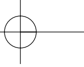
Kullanıcı, canvas elemanı üzerinde Mouse'un sol tuşuna tıkladığında curPositon() isimli fonksiyon çalışacaktır. Amacımız kullanıcının canvas üzerinde tıkladığı noktanın yatay koordinatını bulup ilerleme çubuğunu bu noktaya göre güncellemek ve video akışını kullanıcının canvas üzerinde tıkladığı noktaya göre elde ettiğimiz currentTime değerine getirmektir. curPositon() fonksiyonu çalıştığında pause() metodu ile medya akışını durduracağız. Canvas elemanının sol üst köşesi ile tıklanan nokta arasındaki yatay uzaklığı elde edip, bu değeri xcor değişkeni içerisinde saklıyoruz.

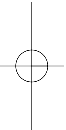
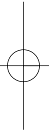
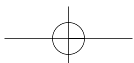
event.offsetX özelliği; olayın oluştuğu elemanın (hedef eleman <canvas>) sol üst köşesi ile kullanıcının tıkladığı nokta arasındaki yatay uzaklığı verir. Fakat offsetX özelliğini desteklemeyen tarayıcılar için event nesnesinin layerX ve canvas elemanının offsetLeft özellikleri kullanılmıştır.

currentTime değerini progress() fonksiyonunda kullandığımız denklemden faydalanarak elde edeceğiz.

```
video.currentTime=(xcor *video.duration)/canvas.width;
```

Daha sonra video akışının belirlediğimiz noktadan itibaren devam etmesi için play() metodunu kullanıp işlemi bitirebiliriz.



SÜRÜKLE-BIRAK İŞLEMLERİ ARAYÜZÜ (DRAG AND DROP API)

9

Sürükle-bırak işlemleri, web sayfalarında sıkça kullanılan bir etkileşim türüdür. HTML5 bildirimiyle tanımlanan **Drag and Drop API** (DnD) arayüzünü kullanarak sürüklenebilir (taşınabilir) nesneler oluşturabilirsiniz. Tarayıcı `DragEvents` (*Sürükleme Olayları*) oluştuğunda **DnD** arayüzünden üretilen `dataTransfer` nesnesini kullanarak sürükleme işlemini gerçekleştirir. Sürükleme işlemi, sürüklenecek kaynak eleman ve bu elemanın bırakılacağı hedef eleman arasında gerçekleştirilir. Aslında sürüklenen kaynak eleman değil bu elemanla ilişkili olan bir veridir.

Özetle sürükle-bırak işlemi; `DragEvents` ve `dataTransfer` nesnesinin özellik ve metotları kullanılarak gerçekleştirilir.

dataTransfer Nesnesi

`dataTransfer` nesnesi, sürüklenecek veriyi saklar ve `event` nesnesi içinde tanımlıdır (sürükleme olaylarında `dataTransfer` nesnesine `event.dataTransfer` şeklinde ulaşacağız). Bu nesne **DnD** arayüzü ile tanımlanan özellik ve metotlara sahiptir. Aşağıda bu nesne için tanımlı olan özellik ve metotlara bakacağız.

dataTransfer Nesnesi Özellikleri

Özellik	Açıklama
`dropEffect`	Sürükle-bırak işleminde kaynak veri, hedef nesne üzerine bırakıldığında kullanılacak tarayıcı davranışını ayarlar. Bu özelliği kullanarak kaynak verinin hedef nesneye kopyalanmasını ya da taşınmasını sağlayabilirsiniz. Bu özelliğin aldığı değer `effectAllowed` özelliği ile tanımlanan değer ya da değerlerden biri olmalıdır. Bu özelliğe `dragenter` ve `dragover` olaylarında değer atanabilir.

<table>
<tr><td></td><td> Kullanımı: <code>dataTransfer.dropEffect[=value]</code> Bu özelliğin alabileceği değerler: <code>copy</code>, <code>move</code>, <code>link</code>, <code>none</code> </td></tr>
<tr><td><code>effectAllowed</code></td><td> <code>dropEffect</code> özelliği tarafından kullanılabilecek, izin verilen bırakma davranışını/davranışlarını ayarlamak için kullanılır. Bu özelliğe <code>dragstart</code> olayında değer atanabilir. Kullanımı: <code>dataTransfer.effectAllowed[=value]</code> Bu özelliğin alabileceği değerler: <code>none</code>, <code>copy</code>, <code>copyLink</code>, <code>copyMove</code>, <code>link</code>, <code>linkMove</code>, <code>move</code>, <code>all</code>, <code>uninitialized</code> Aşağıda <code>effectAllowed</code> ve <code>dropEffect</code> özelliklerine atanabilecek değerler sonucunda tarayıcı tarafından gösterilecek davranışlar listelenmiştir.
<table>
<tr><th>effectAllowed</th><th>dropEffect</th><th>Tarayıcı Davranışı</th></tr>
<tr><td><code>uninitialized</code>, <code>copy</code>, <code>copyLink</code>, <code>copyMove</code>, <code>all</code> değerlerinden herhangi biri</td><td><code>copy</code></td><td><code>copy</code></td></tr>
<tr><td><code>uninitialized</code>, <code>link</code>, <code>copyLink</code>, <code>linkMove</code>, <code>all</code> değerlerinden herhangi biri</td><td><code>link</code></td><td><code>link</code></td></tr>
<tr><td><code>uninitialized</code>, <code>move</code>, <code>copyMove</code>, <code>linkMove</code>, <code>all</code> değerlerinden herhangi biri</td><td><code>move</code></td><td><code>move</code></td></tr>
</table></td></tr>
<tr><td>Diğer Özellikler</td><td><code>files</code>, <code>types</code></td></tr>
</table>

dataTransfer Nesnesi Metotları

Metod	Açıklama
`setData()`	Sürüklenecek veriyi ve türünü tanımlamak için kullanılır. **Kullanımı:** `dataTransfer.setData(format,data)` `format` parametresi ile sürüklenecek verinin türünü tanımlanır. Sürüklenecek veri; bir metin, link, dosya, resim ya da bir HTML düğümü olabilir. Format özelliğine atanabilecek değerlere tarayıcıların verdiği destek farklılık göstermektedir. Şu an için çoğu tarayıcı sadece `text/plain` ve `text/uri-list` değerlerine destek vermektedir. Bu yüzden bölüm boyunca format özelliğine sadece `text/plain` değerini atayarak metin ve eleman sürükleme işlemlerini yapacağız.

	`data` parametresi sürüklenecek veriyi tanımlar. Bu metot, `dragstart` olayında kullanılmalıdır.
`getData()`	Sürüklenen veriyi elde etmek için kullanılır. **Kullanımı:** `data=dataTransfer.getData(format)` `format` parametresi ile `dataTransfer` nesnesi içinde bulunan, yani ulaşmak istediğimiz verinin türü tanımlanır. Eğer bir metin sürükleme işlemi yapılıyorsa, `getData()` metodu içerisinde kullanacağımız `format` değeri `text/plain` olmalıdır. Bu metot `drop` olayında kullanılmalıdır.
Diğer Metotlar	`addElement()`, `setDragImage()`, `clearData()`

DragEvent (Sürükleme Olayları)

Olay	Açıklama
`drag`	Sürükleme işlemi boyunca **kaynak eleman** (sürüklenen eleman) için drag olayı oluşur.
`dragend`	Sürükleme işlemi bittiğinde, yani kullanıcı hedef eleman üzerinde fareyi serbest bıraktığında **kaynak eleman** için `dragend` olayı oluşur.
`dragstart`	Sürükleme işlemi başladığında **kaynak eleman** için `dragstart` olayı oluşur.
`dragleave`	Kaynak eleman **hedef eleman** üzerinden ayrıldığında hedef eleman için dragleave olayı oluşur.
`dragover`	Kaynak eleman **hedef eleman** üzerinde sürüklendiğinde hedef eleman için `dragover` olayı oluşur.
`dragenter`	Kaynak eleman **hedef eleman** üzerine giriş yaptığında hedef eleman için `dragenter` olayı oluşur.
`drop`	Sürükleme işlemi bittiğinde yani kullanıcı hedef eleman üzerinde fareyi serbest bıraktığında **hedef eleman** için drop olayı oluşur.

Şimdi basit bir sürükle bırak uygulaması yaparak işleme başlayalım...

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
```

```
<title>Basit Sürükle-Bırak</title>
<style>
    #drop
    {
        min-height:100px;
        width:200px;
        border:1px solid #cc0066;
        margin:5px;
        padding:10px;
    }
    #drag
    {
        background-color: #990000;
        width:80px;
        padding:5px;
        text-align: center;
        color:White;
        font-size:17px;
        font-weight:bold;
        cursor:move;
    }
</style>
<script type="text/javascript">
    /*Event nesnesi meydana gelen olay(event) ile ilgili ayrıntılı bilgilere ulaşmak ve olay akışını kontrol etmek için kullanılır. DOM Event Interface(Olay Arayüzü) ile tanımlı olan event nesnesini farklı tarayıcılarda doğru olarak elde edebilmek için eventObject isimli bir nesne oluşturacağız. IE'de  event nesnesine window.event, Firefox ve diğer tarayıcılarda sadece event şeklinde ulaşılır.
    */
    var eventObject = {
        getEvent: function (event) {
            return event ? event : window.event;
        },
        getTarget: function (event) {
            return event.target || event.SrcElement;
            /*Olayın meydana geldiği elemana ulaşmak için (Elemanı nesne olarak
            elde etmek için);
            w3c, firefox ve diğer tarayıcılarda>>
            target özelliği kullanılır. target özelliği;
            DOM Level 2 Events ile tanımlanmıştır.
```

```
        Internet Explorer da ise>>
        srcElement özelliği kullanılır.
        */
    },
    getPreventDefault: function (event) {
       /*
        Olayın eleman üzerinde oluşturduğu varsayılan davranışı iptal
        etmek için;
        w3c,firefox ve diğer tarayıcılarda>>
        preventDefault() metodu kullanılır. Bu metod;
        DOM Level 2 Events ile tanımlanmıştır.
        Internet Explorer da ise>>
        returnValue özelliği kullanılır.
        Bu özelliğe false değeri atanmalıdır.
        ////////////////////////////////////////////
        event.preventDefault() metodu olay akışını durdurmaz.
        [Bu metot cancelable=true yani engellenebilir olaylar için kullanılır.]
        */
        if (event.preventDefault) {
            event.preventDefault()
        } else {
            event.returnValue = false;
        }
    }
};

var init = function () {
    var target = document.querySelector('#drop');
    var drag = document.querySelector('#drag');
    /*HTML5 drag and drop API kullanarak en basit anlamda bir sürükleme
    işlemi yapmak için ilk olarak sürüklenecek elemanın draggable özelliğine
    (attribute)  true değeri atanmalıdır.
    Daha sonra bu elemanın dragstart olayına bir olay dinleyicisi eklenmeli
    ve bu olay dinleyici fonksiyon içerisinde sürüklenecek veri tanımlanmalıdır.
    */

    drag.addEventListener('dragstart', function (event) {
        var evt = eventObject.getEvent(event);
        // event nesnesi elde edildi.
        evt.dataTransfer.setData("text", evt.target.id);
        /*dataTransfer nesnesi içerisine sürüklenecek div#drag elemanının
```

```
                    id özelliğinin değerini atıyoruz. ('drag' verisi sürüklenecek)
                    */
            }, false);

            target.addEventListener('drop', function (event) {
                var evt = eventObject.getEvent(event);
                var idDrag = event.dataTransfer.getData('text');
                /*dataTransfer nesnesi içerisindeki veri idDrag isimli değişken
                  içerisine atanıyor.*/
eventObject.getTarget(event).appendChild(document.getElementById(idDrag));
                /*Sonuç olarak id özelliğinin değeri idDrag değişkeni içerisindeki
                  metinle aynı olan eleman hedef eleman içerisine ekleniyor. */

                eventObject.getPreventDefault(event);
            }, false);
            /*Hedef elemanın 'dragover' ve 'dragenter' olayları için aşağıdaki olay
            dinleyicilerini ekleyerek uygulamamızı bitirelim.*/
            target.addEventListener('dragover', function (event) {
                eventObject.getPreventDefault(event);
            }, false);

            target.addEventListener('dragenter', function (event) {
                eventObject.getPreventDefault(event);
            }, false);

        }
    </script>
</head>
<body onload="init();">
    <div id="drag" draggable="true">Sürükle1</div>
    <div id="drop">
    </div>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü.

Yukarıdaki örneğimizde kaynak elemanı (#`drag`) hedef eleman (#`drop`) üzerine bıraktığımızda kaynak eleman `appendChild()` metoduyla (#`drop`) isimli eleman içerisine ekleniyor. Bu ekleme işlemi (#`drop`) elemanın içerisindeki elemanların normal akışına göre yapılıyor. Örneğin; (#`drop`) elemanının içerisinde bir div elemanı olsaydı ve biz taşıma işlemini yapsaydık aşağıdaki ekran görüntüsünü alacaktık.

Aşağıdaki satırı (#`drop`) elemanı içerisine ekleyelim.

```
<div style="background-color:#66CCFF;">drop>>div</div>
```

Sonuç:



Firefox 4 ekran görüntüsü.

Bir sonraki sayfadaki örnekte kullanıcının #`drop` elemanı içerisine sürüklediği resimlerin bu eleman içerisinde birer kopyasını oluşturacağız. Ayrıca bu kopyanın altına resmin `alt` özelliğine aldığı değeri iliştireceğiz.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Resim Kopyala(Sürükle-Bırak)</title>
    <style>
        #drop
        {
            width: 64px;
            padding: 2px 12px;
            height: 300px;
            border: 1px solid #cc0066;
            float: left;
            margin-left: 15px;
        }
        #drag
        {
            float: left;
            width: 64px;

        }
    </style>
    <script type="text/javascript">
        var eventObject = {
            getEvent: function (event) {
                return event ? event : window.event;
            },
            getTarget: function (event) {
                return event.target || event.SrcElement;
            },
            getPreventDefault: function (event) {

                if (event.preventDefault) {
                    event.preventDefault()
                } else {
                    event.returnValue = false;
                }
            }
        };
```

```
        var init = function () {
            var target = document.querySelector('#drop');
            var drag = document.querySelectorAll('#drag img');
            /*
            document.querySelector() metodu seçici ile eşleşen tüm elemanların
bir listesini döndürür. Bu liste StaticNodeList (Static düğüm listesi) türündendir.
            */

            for (var i = 0; i < drag.length; i++) {
                drag[i].setAttribute('draggable', 'true');
            }
            /*#drag elemanı içerisindeki tüm img elemanlarına draggable
           özelliğini ekledik*/
            for (var i = 0; i < drag.length; i++) {
                drag[i].addEventListener('dragstart', dragStart, false);
            }
            /*#drag elemanı içerisindeki tüm img elemanlarının dragStart olaylarına
            olay dinleyicisi ekledik.*/
            target.addEventListener('drop', function (event) {
                var evt = eventObject.getEvent(event);
                var data = event.dataTransfer.getData('text');
                evt.target.innerHTML += data;
                /*dataTransfer nesnesi içerisindeki metin innerHTML özelliği
                  kullanılarak #drop elemanı içerisine eklendi.
                */
                eventObject.getPreventDefault(event);
            }, false);
            target.addEventListener('dragover', function (event) {
                eventObject.getPreventDefault(event);
            }, false);

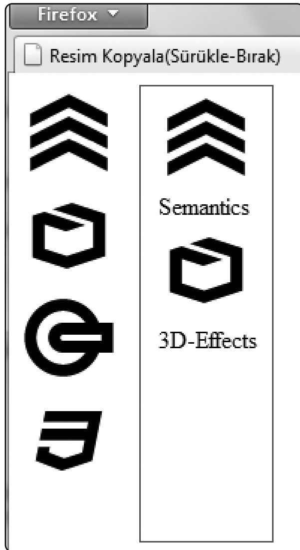
            target.addEventListener('dragenter', function (event) {
                eventObject.getPreventDefault(event);
            }, false);

        }
        var dragStart = function (event) {
            var evt = eventObject.getEvent(event);
            evt.dataTransfer.setData('text', "<img src='" +
evt.target.src + "'/><br/><span>" + evt.target.alt + "</span>");
            /*dataTransfer nesnesi içerisinde metin saklayarak resim kopyalama işlemi
yapacağız. dataTaransfer nesnesi içerisinde; HTML olarak bir img birde span elemanı
```

```
oluşturacak formatta yazılan bir metin saklayacağız. Sürüklenen resmin src özelliğinin değeri
metin olarak yazdığımız img elemanının src özelliğine  atanacaktır. Ayrıca sürüklenen resmin
alt özelliğinin değeri metin olarak formatladığım span elemanı içerisine eklenecektir.*/
          }
     </script>
</head>

<body onload="init();">
     <div id="drag">
          <img src="HTML5_Semantics_64.png"  alt="Semantics"/>
          <img src="HTML5_3D_Effects_64.png" alt="3D-Effects"/>
          <img src="HTML5_Offline_Storage_64.png" alt="offline-Storage"/>
          <img src="HTML5_Styling_64.png" alt="Style"/>
     </div>
     <div id="drop">
     </div>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü.

GEOLOCATION API

10

Geolocation API kullanarak geliştireceğiniz web uygulamalarıyla kullanıcıların konum (coğrafik koordinat) bilgilerine ulaşabilir ve bu bilgileri kullanıcıların hizmetine sunabilirsiniz (Kullanıcının konumunu paylaşmayı kabul etmesi şartıyla). Elde edilen konum bilgisi **Enlem** ve **Boylam** verisi şeklinde olacaktır. Elde edilen konum bilgisini **online harita servisleri** ile beraber kullanıp, kullanıcıya konumunu görsel olarak (harita üzerinde) göstermek işin en faydalı taraflarından biridir.

GEOLOCATION NESNESİ

Coğrafik konum bilgisini elde etmek için kullanılan temel nesnedir. **Geolocation** nesnesi `window.navigator` nesnesinin alt nesnesi olarak tanımlanmıştır. Tarayıcıların eski sürümlerinin Geolocation API desteklememe durumunu dikkate alarak kodlarınızı aşağıdaki gibi bir **if** yapısı içerisine yazmayı unutmayınız.

```
if (navigator.geolocation) {
   /*geolocation API kullanılabilir*/
} else {
   /*geolocation API desteklenmiyor*/
}
```

Şimdi bu nesnenin metotlarına bakalım.

getCurrentPosition()

Tarayıcının çalıştığı bilgisayarın geçerli konumunu elde etmek için kullanılan metottur.

Kullanımı:

navigator.geolocation.getCurrentPosition(successCallback[,errorCallback][,options])

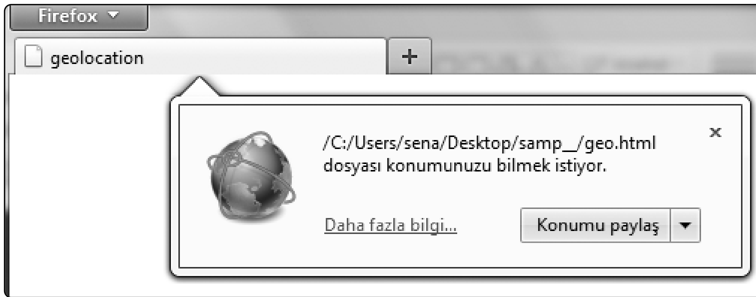
- **successCallback** (gerekli) parametresi position nesnesini parametre olarak kabul eden bir fonksiyon tanımlar. Bu fonksiyon içerisinde enlem ve boylam bilgileri elde edilir.
- **errorCallback**, coğrafik konum bilgisi elde edilirken bir hata meydana geldiğinde (işlem başarısız olduğunda) çalışacak fonksiyonu tanımlar. Oluşan hata ile ilgili bilgi elde etmek için bu fonksiyona bir parametre tanımlaması yapılmalıdır. (Tanımlanan parametre PositionError tipinde bir nesne olarak işlem görür.)
- **options** parametresi pozisyon hesaplanırken kullanılacak PositionOptions nesnesinin özelliklerine değer atamak için kullanılır.
- **geolocation** nesnesi ile beraber kullanılan position, PositionError, PositionOptions nesneleri ayrıntılı olarak incelenecektir.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>geolocation</title>
    <script type="text/javascript">
        var init = function () {
            if (navigator.geolocation) {
                navigator.geolocation.getCurrentPosition(successFunction,
errorFunction);
            } else {
                alert('Geolocaiton API tarayıcı tarafından
desteklenmiyor');
            }
        }
        var successFunction = function (position) {
            var lat = position.coords.latitude; /*Enlem*/
```

```
            var lng = position.coords.longitude; /*Boylam*/
            console.log('Enlem(latitude):' + lat + '\nBoylam(longitude):'
+ lng);
        }
        var errorFunction = function (err) {
            alert('hata!');
        }
    </script>
</head>
<body onload="init();">
</body>
</html>
```

Tarayıcıyı çalıştırdığımızda ilk önce konum bilgisini paylaşmak isteyip istemediğimiz sorulacaktır.



Konumu paylaş deyip **firebug** konsoluna baktığınızda konumunuzu sayısal olarak elde ettiğinizi göreceksiniz.

watchPosition()

`getCurrentPosition()` metodu ile aynı parametreleri alan bu metodun farkı, konum bilgilerini güncelleyebilmesidir. Elde edilmiş konum bilgisinden daha doğru bir konum bilgisine ulaşıldığında ya da kullanıcının pozisyonu değiştiğinde konum bilgileri güncellenir (`position` nesnesi içerisindeki veriler güncellenir). Özellikle mobil cihazlarda `getCurrentPosition()` yerine bu metodu tercih edebilirsiniz.

Kullanımı:

```
watchId = navigator.geolocation.watchPosition(successCallback
[, errorCallback] [, options])
```

watchId değeri clearWatch() metodu ile kullanılmak üzere watchPosition() metodu tarafından döndürülen benzersiz bir kimlik değeridir.

clearWatch()

watchPosition() metodu ile ilişkili olarak kullanılan bu metot coğrafik konum güncellemelerini durdurmak için kullanılır. Yani bu metot kullanıldıktan sonra kullanıcının coğrafik konum değişiklikleri izlenmez ve konum bilgileri güncellenmez.

Kullanımı: **navigator.**geolocation.clearWatch(watchId)

position Nesnesi

Geolocation nesnesinin getCurrentPosition() ve watchPosition() metotlarının successCallback parametresini hatırlayınız. Bu parametre yerine bir fonksiyon tanımlıyorduk. İşte bu fonksiyon içerisinde konum bilgilerine ulaşmak için position nesnesi kullanılır.

Özellik	Açıklama
coords	getCurrentPosition() ve watchPosition() metotları tarafından elde edilen coğrafik konum bilgilerini saklayan nesnedir.
timestamp	Konum bilgisinin elde edilme zamanını döndürür.

coords Nesnesi

Aşağıdaki özellikler **readonly** (*sadece okunabilir*)'dir. Bu özelliklerin aldığı değerlere ulaşmak için position.coords.özellik şeklinde bir yol izlenmelidir.

Özellik	Açıklama
latitude	Ondalık derece cinsinden **enlem** verisini elde etmek için kullanılır. Bu değer -90.00 ile +90.00 arasında olabilir.
longitude	Ondalık derece cinsinden **boylam** verisini elde etmek için kullanılır. Bu değer -180.00 ile +180.00 arasında olabilir.
altitude	Rakım bilgisini (yükseklik) elde etmek için kullanılır. Bu veri **WGS 84** (*World Geodetic System*) sistemi kullanılarak elde edilir.
Diğer Özellikler	heading, speed, accuracy, altitudeAccuracy

positionError Nesnesi

getCurrentPosition() ve watchPosition() metotlarının errorCallback parametresi, hata olduğunda çalışacak fonksiyonu tanımlıyordu. İşte bu fonksiyon içerisinde oluşan hata ile ilgili bilgi elde etmek için positionError nesnesinin özellikleri kullanılır. errorCallback fonksiyonu için yazmış olduğunuz parametre, fonksiyon içerisinde positionError nesnesi olarak işlem görür.

Örnek:

```
var errorFunction = function (error) {
    alert(error.code);
    // error parametresi positionError nesnesini temsil eder.
}
```

Aşağıdaki özellikler readonly'dir.

Özellik	Açıklama
code	Konum bilgisi elde edilirken meydana gelen hatanın türünü gösteren bir tamsayı döndürür. Aşağıda code özelliğinin alabileceği sayısal değerler (eşdeğer hata kodları) listelenmiştir. **UNKNOWN_ERROR (sayısal değeri 0)** Aşağıdaki hata durumları dışında bilinmeyen bir hata meydana gelmiştir. **PERMISSION_DENIED (sayısal değeri 1)** Kullanıcı konumunu paylaşmayı kabul etmemiştir. **POSITION_UNAVAILABLE (sayısal değeri 2)** Kullanıcının konumu tespit edilememiştir. **TIMEOUT (sayısal değeri 3)** Zaman aşımı meydana gelmiştir.
message	Hata ayıklama işlemleri için kullanılabileceğiniz bir değer tanımlaması yapabilirsiniz.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
```

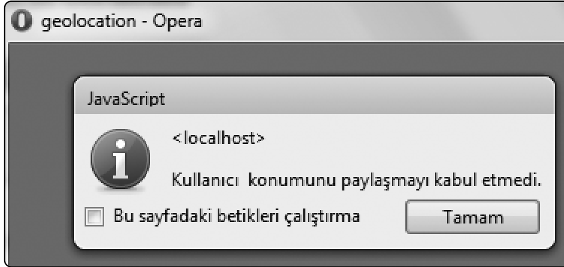
```
    <title>geolocation positionError  Nesnesi </title>
    <script type="text/javascript">
        var init = function () {
            if (navigator.geolocation) {
                navigator.geolocation.getCurrentPosition(successFunction,
errorFunction);
            } else {
                alert('Geolocaiton API tarayıcı tarafından
desteklenmiyor');
            }
        }
        var successFunction = function (position) {
            var lat = position.coords.latitude; /*Enlem*/
            var lng = position.coords.longitude; /*Boylam*/
            alert('Enlem(latitude):' + lat + '\nBoylam(longitude):' + lng);
           alert(position.altitude);
        }

       var errorFunction = function(error) {
           var msg = 'Hata !';
            switch (error.code) {
                case error.UNKNOWN_ERROR:
                    msg= "Blinmeyen bir hata oluştu";
                    break;
                case error.PERMISSION_DENIED:
                    msg = "Kullanıcı  konumunu paylaşmayı kabul etmedi.";
                    break;
                case error.POSITION_UNAVAILABLE:
                    msg = "Pozisyon bilgisi elde edilemedi";
                    break;
                case error.TIMEOUT:
                    msg = "zaman aşımı gerçekleşti";
                    break;
            }
            alert(msg);
        }
    </script>
</head>
<body onload="init();">
</body>
</html>
```

Ekran görüntüsü:



Reddet düğmesine tıklayıp sonuca bakalım.



Opera 11 ekran görüntüsü

positionOptions Nesnesi

`getCurrentPosition()` ve `watchPosition()` metotları için `options` parametresini hatırlayınız. `Options` parametresi, `positionOptions` nesnesinin özelliklerine değer atamak için kullanılmaktaydı. `positionOptions` nesnesinin özellikleri aşağıda verilmiştir.

Özellik	Açıklama
enableHighAccuracy	Boolean türünden bir özelliktir (true ya da false değerlerinden birini alır). En iyi konum bilgisini elde etmek için kullanabileceğiniz bir özelliktir. Fakat bu özelliğin kullanılması durumunda geç yanıt alma ya da fazla güç tüketimi gibi sorunlar ortaya çıkabilir. Bu özelliğin varsayılan değeri false'dur.
timeout	Tarayıcının konum verisini elde etmek için kullanabileceği maksimum süreyi (izin verilen süre) tanımlar. Bu özelliğe milisaniye cinsinden bir değer atanmalıdır.
maximumAge	Önbelleğe alınan konum bilgisi için zaman aşımı süresini belirler.

Bu özelliğe milisaniye cinsinden bir değer atanmalıdır. Bu özelliklere aşağıdaki şekilde değer atayabilirsiniz.

```
navigator.geolocation.getCurrentPosition(successFunction, errorFunction,
{ enableHighAccuracy: true, timeout:17000,maximumAge: 20000 });
```

Online Harita Servislerini Kullanmak

Geolocation nesnesini kullanarak elde edeceğimiz bilgiler sayısal olacaktır. Bu durumda bu sayısal konum bilgileri ile beraber online harita servislerini kullanıp kullanıcıya konumunu görsel olarak harita üzerinde gösterebiliriz.

Online harita servisi olarak **Google Maps** ya da **OpenStreetMap/OpenLayers** kullanabilirsiniz. Fakat kullanım kolaylığı ve verdiği gelişmiş harita desteği açısından **Google Maps** kullanmayı tercih edeceğiz.

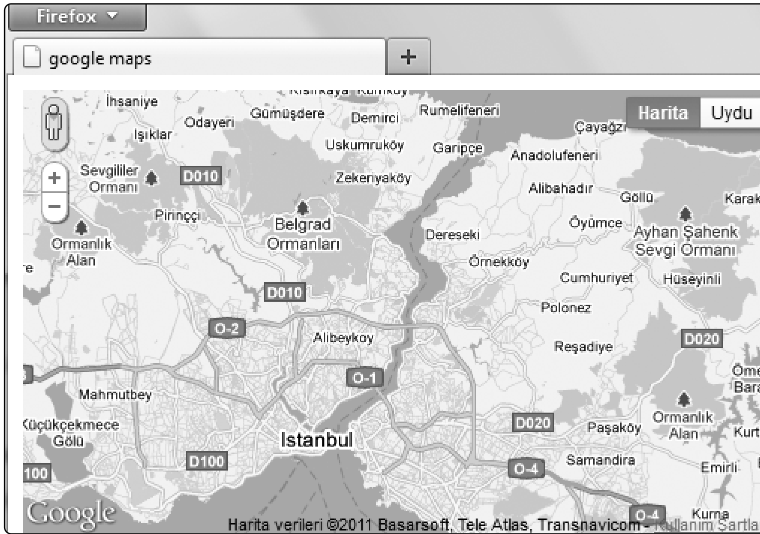
İlk önce enlem ve boylam bilgilerini sayısal olarak girerek online harita servislerinin kullanımına bakalım.

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>google maps</title>
    <style type="text/css">
        html
        {
            height: 100%;
        }
        body
        {
            height: 100%;
            margin: 0px;
            padding: 10px;
        }
    </style>
    <script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=true">
    </script>
    <script type="text/javascript">
        function init() {
            var latlng = new google.maps.LatLng(41.10, 29.05);
            var mapOptions = {
                zoom: 10,
                center: latlng,
```

```
                    mapTypeId: google.maps.MapTypeId.ROADMAP
                };
                var map = new google.maps.Map(document.getElementById("map"),
            mapOptions);
            }
        </script>
</head>
<body onload="init()">
        <div id="map" style="width:400px; height: 200px">
        </div>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Google Maps metotlarını kısaca anlatmak için yukarıdaki uygulamada enlem ve boylam değerlerini kendimiz girdik (HTML5 Geolocation API kullanmadık). **Google Maps** servisini kullanmak için ilk önce <head> etiketleri arasına aşağıdaki <script> etiketini ekledik (Google Maps API'yi sayfaya dahil etmek için).

```
<script type="text/javascript"
      src="http://maps.google.com/maps/api/js?sensor=true">
</script>
```

Enlem ve boylam değerlerini saklayacak bir `Latlng` nesnesi tanımlamamız gerekir. `LatLng` nesnesi oluşturmak için aşağıdaki yol izlenmelidir.

```
var Latlng = new google.maps.LatLng(Latitude, Longitude)
```

Latitude: Enlem.

Longitude: Boylam.

Enlem ve boylam değerlerini girerek `LatLng` nesnesi oluşturduk.

```
var latlng = new google.maps.LatLng(41.10, 29.05);
```

Harita ile ilgili seçenekleri tanımlamak için `mapOptions` isimli bir nesne oluşturduk.

```
var mapOptions = {
      zoom: 10,
      center: latlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
};
```

zoom: Yakınlaştırma oranını ayarlamak için kullanılır.

center: Bu özelliğe latlng nesnesi atanır (Hedef konum).

mapTypeId: Kullanılacak harita türünü tanımlar.

Kullanabileceğiniz harita türleri de şöyledir:

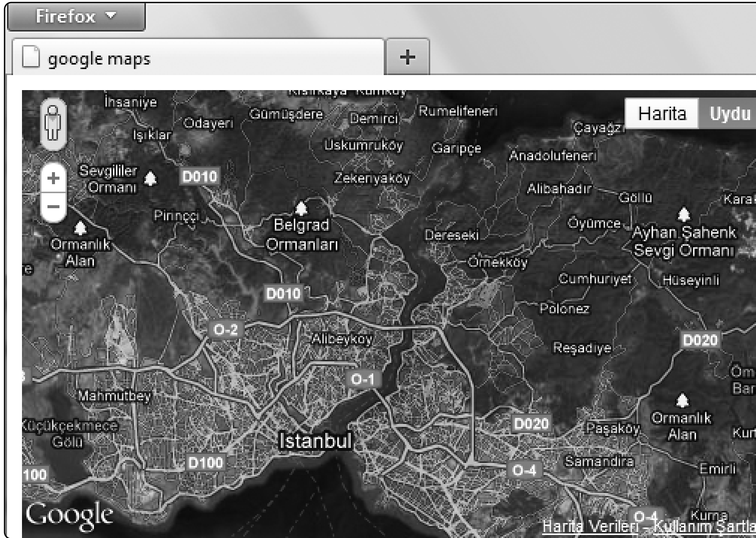
```
google.maps.MapTypeId.ROADMAP
google.maps.MapTypeId.SATELLITE
google.maps.MapTypeId.HYBRID
google.maps.MapTypeId.TERRAIN
```

Son olarak `mapOptions` nesnesi içerisindeki özellikleri kullanan ve `div#map` elemanı içerisinde gösterilecek bir `map` nesnesi tanımlayarak işlemi bitiriyoruz.

```
var map = new google.maps.Map(document.getElementById("map"), mapOptions);
```

Yukarıdaki uygulamadaki `mapOptions` nesnesi içerisinde tanımlanan `mapTypeId` değerini aşağıdaki gibi değiştirerek uygulamayı tekrar çalıştıralım.

```
var mapOptions = {
    zoom: 10,
    center: latlng,
    mapTypeId: google.maps.MapTypeId.HYBRID
};
```



Firefox 4 ekran görüntüsü

Şimdi **HTML5 Geolocation API + Google Maps** kullanarak bulunduğumuz konumu bize, harita üzerinde görsel olarak gösterecek bir örnek uygulama geliştireceğiz.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset=”utf-8” />
    <title>google maps</title>
    <style type=”text/css”>
        body
        {
            height: 100%;
            margin: 0px;
            padding: 10px;
        }
    </style>
    <script type=”text/javascript”
src=”http://maps.google.com/maps/api/js?sensor=true”>
    </script>
    <script type=”text/javascript”>
        function init() {
            if (navigator.geolocation) {
```

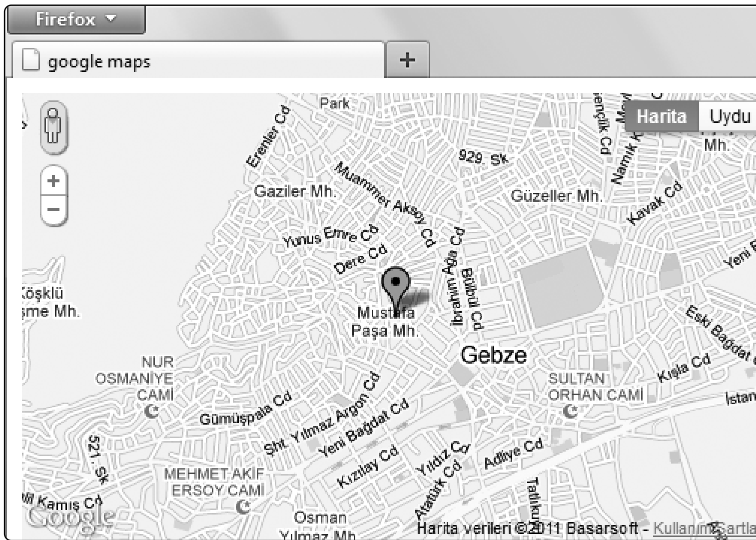
```
            /*Eğer geolocation nesnesi tarayıcı tarafından destekleniyor ise
            getCurrentPosition() metodu çağrılacak.
            */
           navigator.geolocation.getCurrentPosition(pozisyonBul,hata);
        } else {
            alert('Tarayıcı geolocation API desteklemiyor....');
        }
    }
    var pozisyonBul = function (position) {
        /*Bulunduğumuz konumun enlem ve boylam bilgilerini
        coords nesnesini kullanarak elde edeceğiz.*/
        var lat = position.coords.latitude;
        var lng = position.coords.longitude;
        /*Enlem ve boylam değerlerini saklayan bir latlng nesnesi oluşturalım.*/
        var latlng = new google.maps.LatLng(lat, lng);
        /*Harita özelliklerini tanımlayalım*/
        var mapOptions = {
            zoom: 14,
            center: latlng,
            mapTypeId: google.maps.MapTypeId.ROADMAP
        };
       /*Sonuç olarak haritanın div#map elemanı içerisinde gösterilmesini sağlayalım*/
        var map = new google.maps.Map(document.getElementById("map"),
    mapOptions);

        /*Son olarak basit bir şekilde bulunduğumuz noktayı harita üzerinde
          işaretleyecek bir baloncuk oluşturarak işlemi bitirelim*/
            var marker = new google.maps.Marker({
            position: latlng,
            map: map,
            title: "Buradasınız"
        });
    }

    var hata = function (error) {
        /*Konum bilgisi hesaplanırken hata oluşursa çalışacak kodlar...*/
        var msg = 'Hata !';
        switch (error.code) {
            case error.UNKNOWN_ERROR:
                msg = "Blinmeyen bir hata oluştu";
                break;
```

```
                case error.PERMISSION_DENIED:
                    msg = “Kullanıcı  konumunu paylaşmayı kabul
etmedi.”;
                    break;
                case error.POSITION_UNAVAILABLE:
                    msg = “Pozisyon bilgisi elde edilemedi”;
                    break;
                case error.TIMEOUT:
                    msg = “zaman aşımı gerçekleşti”;
                    break;
            }
            alert(msg);
        }
    </script>
</head>
<body onload=”init()”>
    <div id=”map” style=”width: 500px; height: 300px”>
    </div>
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 ekran görüntüsü

Tarayıcı Desteği

Geolocation API için tarayıcı destek durumu şöyledir:

Tarayıcı	Sürümü
Internet Explorer	IE9+
Firefox (Gecko)	3.5 (1.9.1)+
Opera	10.60
Safari (WebKit)	5 (533)

WEB STORAGE 11

Web Storage (*DOM Storage olarak da adlandırılır*) API, istemci tarafında veri saklama işlemini oldukça basit, işlevsel ve güvenli bir hale getirmektedir. Web storage yapısında veriler (anahtar = değer) şeklinde birer ikili çift olarak saklanır. Web Storage, istemci tarafında veri saklama işlemleri için **sessionStorage** ve **localStorage** olarak tanımlanan iki ayrı yapı sunar. İstemci tarafında veri saklamak denince hemen aklınıza **çerezler** (*cookies*) gelebilir.

Web Storage yapısının çerezlerden birçok üstünlüğü bulunmaktadır.

Çerezler ile saklanacak maksimum veri büyüklüğü `4 KB` iken, Web Storage ile `MB`'lar seviyesinde veri saklanabilir. Çerezler de aynı web sitesini örneğin; iki farklı sekmede ya da pencerede çalıştırdığınızda kullanılacak çerez aynı olacağından, problemler yaşanabilmektedir. Bu sorun **sessionStorage** (*oturum depolama*) yapısı ile yaşanmaz. Çünkü bu yapıda veriler oturum bazında saklanır.

sessionStorage ve **localStorage** nesneleri ortak olarak Storage arayüzü ile tanımlanan metot ve özelliklere sahiptirler. Bu nesneler **window** nesnesinin birer özelliği olarak tanımlıdırlar. Örneğin; **sessionStorage** nesnesine `window.sessionStorage` ya da sadece `sessionStorage` yazarak ulaşabilirsiniz.

sessionStorage (Oturum Depolama)

Veriler sadece oturum bazında saklanır. Yani kullanıcı bir web sitesini açtığında `sessionStorage` ile saklanan veri sadece belirtilen web sitesinin açıldığı tarayıcı

penceresi ya da sekmesi için geçerli olur. Sekme ya da pencere kapatıldığında `sessionStorage` ile saklanan veriler silinir.

Kullanımı: `window.sessionStorage.(Özellik/Metot)`

localStorage (Yerel Depolama)

İstemci tarafında oturumdan bağımsız, kalıcı olarak veri saklamak için kullanılır. Yerel veri sadece kendisini oluşturan web sitesi tarafından kullanılabilir (Veri saklandığı süre boyunca). Saklanacak veriler için bir zaman sınırlaması yoktur.

`window.localStorage.(Özellik/Metot)`

Storage Nesnesi (sessionStorage, localStorage) Özellik ve Metotları

`sessionStorage` ve `localStorage` nesneleri Storage arayüzü ile tanımlanan aşağıdaki ortak özellik ve metotlara sahiptirler.

setItem()

İkili bir veri çifti oluşturmak için kullanılır. Bu veri çifti bir anahtar adı birde değerden oluşur. Kısacası; bir değişken ve bu değişken için bir değer tanımlar.

Kullanımı:

```
storage.setItem('key', 'değer');
storage['key'] = 'değer';
storage.key = 'değer';
```

Örnek:

```
window.localStorage.setItem('tarayıcı','firefox 4');
```

getItem()

Belirtilen anahtarın (değişken adı) değerini elde etmek için kullanılır.

Kullanımı: `deger = storage.getItem('key')`

Örnek:

```
localStorage.setItem('tarayıcı','firefox 4');
alert(localStorage.getItem('tarayıcı'));
```

REMOVEITEM()

Belirtilen anahtarı değeri ile beraber siler.

Kullanımı: `storage.removeItem('key')`

Örnek:

```
localStorage.setItem('tarayıcı','firefox 4');
alert(localStorage.getItem('tarayıcı')); // sonuç :firefox 4
localStorage.removeItem('tarayıcı')
alert(localStorage.getItem('tarayıcı')); // sonuç:null
```

CLEAR()

Tüm veri çiftlerini (anahtarlar ve değerleri) siler.

Kullanımı: `storage.clear()`

KEY()

Indeks numaralarını kullanarak anahtar isimlere ulaşmaya yarar. Indeks numaraları `0`'dan başlayacaktır.

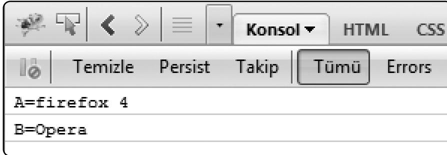
Kullanımı: `storage.key(index)`

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>storage.key()</title>
    <script type="text/javascript">
        var init = function () {
        var storage=window.localStorage;
            storage.setItem('A', 'firefox 4');
            storage.setItem('B', 'Opera');
                var key1 = storage.key(0);
                var value1 = storage.getItem(key1);
                var key2 = storage.key(1);
                var value2 = storage.getItem(key2);
                console.log(key1 + '=' +value1);
                console.log(key2 + '=' +value2);
            }
    </script>
```

```
</head>
<body onload="init();">
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 Firebug ekran görüntüsü

LENGTH ÖZELLİĞİ

`storage` nesnesi ile erişilebilecek veri çiftlerinin (anahtar = değer) sayısını elde etmek için kullanılır.

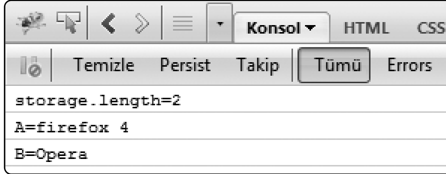
Kullanımı: `storage.length`

Örnek:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>storage.length</title>
    <script type="text/javascript">
        var init = function () {
            var storage = window.localStorage;
            storage.clear();
            /*Daha önce kaydedilmiş yerel veri çiftlerini
              clear() metoduyla siliyoruz.*/
            storage.setItem('A', 'firefox 4');
            storage.setItem('B', 'Opera');
            console.log('storage.length='+storage.length);
            for (var i = 0; i < storage.length; i++) {
                var key = storage.key(i);
                var value = storage.getItem(key);
                console.log(key + '=' + value);
            }
        }
```

```
    </script>
</head>
<body onload="init();">
</body>
</html>
```

Ekran görüntüsü:



Firefox 4 Firebug ekran görüntüsü

NOT

HTML 5 Web Storage yapısı içerisinde `storage` olayını tanımlamıştır. Bu olay `storage` nesnesi tarafından kullanılabilen değer çiftlerinde bir değişiklik olduğu zaman oluşur. `storage` olayında `event` nesnesi aşağıdaki ek özelliklere sahip olacaktır.

- **event.key:** Güncellenen anahtar adını döndürür.
- **event.newValue:** Güncellenen anahtar değerini döndürür.
- **event.oldValue:** Güncellenen anahtarın bir önceki değerini döndürür.
- **event.storageArea:** Olay ile ilintili storage nesnesini döndürür.
- **event.url:** Yerel ya da oturum depolama ile saklanan değer çiftlerinin güncellendiği sayfanın URL adresini döndürür.

Aşağıdaki şekilde storage olayına olay dinleyicisi ekleyebilirsiniz;

```
window.addEventListener("storage", onStorage, false);
var onStorage = function (event) {
    var event = event || window.event;
    // event.key, event.newValue…
}
```

Örnek: `sessionStorage` ve `localStorage` nesnelerinin kullanımını göstermek için basit bir uygulama yapalım.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Storage</title>
```

```
<style>
    tr
    {
        font-family: Calibri;
        font-size: 14px;
        text-indent: 5px;
    }
    table td:nth-of-type(1)
    {
        /*<tr> elemanları içerisindeki ilk <td> elemanları hedef alınmıştır.*/
        background-color: gray;
        width: 100px;
        float: left;
        color: White;
    }
    table td:nth-of-type(2)
    {
        /*<tr> elemanları içerisindeki ikinci <td> elemanları hedef alınmıştır.*/
        background-color: #cccccc;
        width: 500px;
        float: right;
    }
    caption
    {
        display: block;
        line-height: 1.5em;
        font-family: Verdana;
        background-color: #ffcc99;
    }
</style>
<script type="text/javascript">

    var init = function () {
        domStorageGet();
        /*sayfa yüklendiğinde localStorage ve sessionStorage verilerinin sayfada
          gösterilmesi için domStorageGet() fonksiyonu çağrılmıştır. Sayfanın tarayıcı
          penceresine ilk yüklenmesinde herhangi bir sessionStorage verisi olmaz.
        */

    }
    var domStorageGet = function () {
```

local Storage..................

```
try
{ document.body.removeChild(document.getElementById('local')); } catch (e) {
}
```

/*Sayfa tarayıcıya ilk yüklendiğinde body elemanı içerisinde table#local elemanı yoktur. Bu eleman(table#local) localStorage nesnesi ile kullanılabilecek bir veri çifti(key,value) olduğunda programatik olarak oluşturacaktır.

domStorageGet() fonksiyonu;

[1] init() fonksiyonu içerisinden çağrılmış ise bu durumda önceden oluşturduğumuz yerel verilere ulaşıp bu verileri programatik olarak oluşturacağımız table#local elemanı içerisinde göstereceğiz.

[2] 'localStorage !' etiketli buton elemanı tarafından çalıştırılmış ise bu durumda kullanıcı yeni yerel bir veri oluşturmuştur. Yapmamız gereken şey table#local elemanını silip kullanıcının oluşturduğu yeni değer çiftini gösterecek şekilde table#local elemanını yeniden oluşturmaktır.

[2] durumu dikkate alarak;

try { document.body.removeChild(document.getElementById('local')); } catch (e) {
}
satırını ekledik.

{removeChild() metodu Dom Level 1,2 core ile tanımlanmıştır}

*/

```
var lStrLength = localStorage.length;
```

/*localStorage nesnesinin kullanabileceği değer çiftlerinin sayısını lStrLength değişkeni içerisine atıyoruz.*/

```
if (lStrLength > 0) {
```

/*Eğer localStorage nesnesi ile ulaşabileceğimiz en az bir değer çifti varsa, bu durumda bu değer çiftinin/çiftlerinin oluşturacağımız tablo içerisinde gösterilmesini sağlayacağız.*/

```
var table = document.createElement('table');
table.id = 'local';
var caption = table.createCaption();
```

/*createCaption() metodu table#local elemanına iliştirilecek
Bir başlık elemanı oluşturur. (<caption></caption>)
{createCaption() metodu Dom Level 2 -HTML bildirimi ile tanımlanmıştır}
*/

```
caption.innerText = 'localStorage';
```

/*caption elemanı içerisine 'localStorage' metnini ekliyoruz

```
<caption>localStorage</caption>
*/
 for (var i = 0; i < lStrLength; i++) {
   /*Bu for döngüsü ile localStorage nesnesinin kullanabileceği
     verileri (yani daha önce bu web belgesi tarafından kaydedilmiş yerel
     verileri) oluşturacağımız tablo içerisinde göstereceğiz.*/

     var key = localStorage.key(i);
     var data = localStorage.getItem(key);
     var row = document.createElement('tr');
     var col1 = document.createElement('td');
     var col2 = document.createElement('td');
     row.appendChild(col1); row.appendChild(col2);
     table.appendChild(row);
     col1.innerText = key;
     col2.innerText = data;
     document.body.appendChild(table);
   /*element.appendChild(childElement) metodu; Bir eleman düğümü
   içerisine kardeş eleman listesinde en sonda olacak şekilde yeni bir
   çocuk eleman ekler. {appendChild() metodu DOM Level 3 Core
   bildirimi ile tanımlanmıştır.}*/
 }
}
// session Storage...
try
{ document.body.removeChild(document.getElementById('session')); } catch (e) {
}
var sStrLength = sessionStorage.length;
if (sStrLength > 0) {
     var table = document.createElement('table');
     table.id = 'session';
     var caption = table.createCaption();
     caption.innerText = 'sessionStorage';

   /*Yukarıdaki satırlar localStorage kodları ile benzer mantıkla
     çalışmaktadır. Fakat sessionStorage nesnesi ile oturum bazında
     saklanan değerlere ulaşmak ve bu değerleri tablo içerisine koymak
     için aşağıda daha farklı bir yol izlenmiştir. İnceleyiniz... */

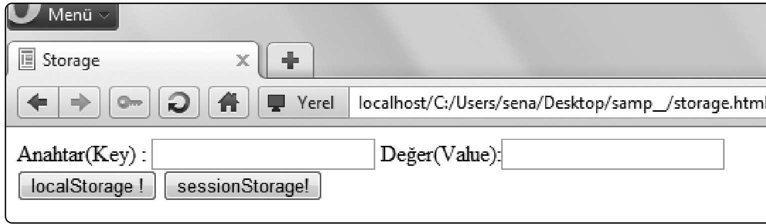
     for (var i = 0; i < sStrLength; i++) {
          var data = [sessionStorage.key(i),
sessionStorage.getItem(sessionStorage.key(i))];
```

```
          var row = document.createElement('tr');
          for (var j = 0; j < 2; j++) {
              /*Bu döngü iki tane td elemanı oluşturma ve
            1. td elemanına key(anahtar adını)
            2.td elemanı içerisine value(değeri)
            yazdırmak için kullanıldı*/
              var col = document.createElement('td');
              col.innerText = data[j];
              row.appendChild(col);
          }
          table.appendChild(row);
          document.body.appendChild(table);
      }
  }

}
/*Aşağıdaki fonksiyonlar;
Kullanıcının metin kutularına girdiği verilere göre yeni bir localStorage ya da
sessionStorage verisi oluşturmak için yazılmıştır. Ayrıca bu fonksiyonlar metin
kutularının kullanıcı tarafından doldurulup doldurulmadığı kontrol edilmektedir.
Son olarak veri çifti oluşturulduktan sonra tabloların güncellenmesi için
domStorageGet() fonksiyonu çağrılmıştır.*/
var lStrSet = function () {
    var key = document.querySelector('input:nth-of-type(1)').value;
    var value = document.querySelector('input:nth-of-type(2)').value;
    if (key == '' || value == '') {
        alert('alanları doldurunuz');
        return;
    }
    localStorage.setItem(key, value);
    domStorageGet();
}
var sStrSet = function () {
    var key = document.querySelector('input:nth-of-type(1)').value;
    var value = document.querySelector('input:nth-of-type(2)').value;
    if (key == '' || value == '') {
        alert('alanları doldurunuz');
        return;
    }
    sessionStorage.setItem(key, value);
```

```
                domStorageGet();
            }
        </script>
    </head>
    <body onload="init();">
        Anahtar(Key) :
        <input type="text" />
        Değer(Value):<input type="text" /><br/>
        <button onclick="lStrSet();">
            localStorage !</button>
        <button onclick="sStrSet();">
            sessionStorage!</button>
    </body>
    </html>
```

Ekran görüntüsü:

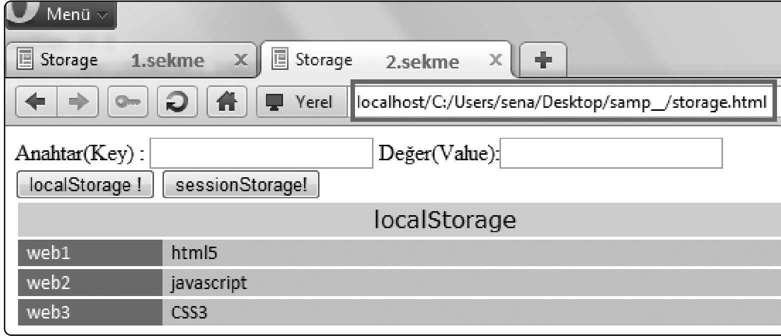


`localStorage` ve `sessionStorage` verisi olarak saklanacak değerleri girip sayfamızın ekran görüntüsüne tekrar bakalım.



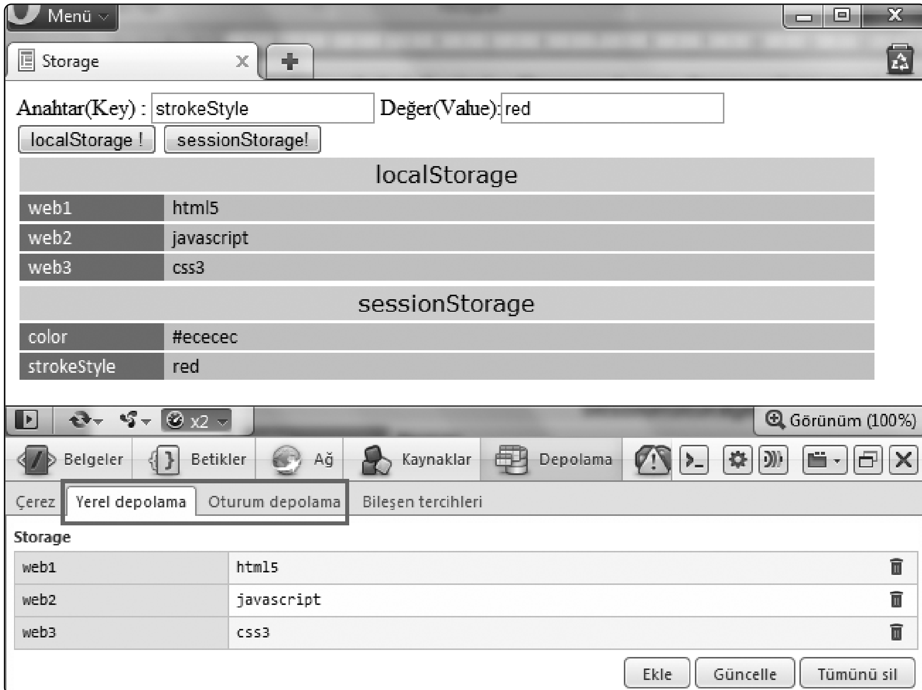
Opera 11 ekran görüntüsü

Metin kutularına girilen değer çifti `localStorage` ya da `sessionStorage` verisi olarak saklanabilir. Aynı sayfayı bir başka sekmede (ya da pencerede) açtığınızda `sessionStorage` verisi olarak saklanan değerleri göremezsiniz. Çünkü bu değerler oturum bazında saklanır.



Opera 11 ekran görüntüsü

Web sayfası tarafından kullanılan `localStorage` ya da `sessionStorage` verilerini görmek ve bu veriler ile ilgili işlemler (silme, güncelleme, ekleme) yapmak için Opera'nın **Dragonfly** aracını kullanabilirsiniz.



Opera 11 ekran görüntüsü

Tarayıcı Desteği

Web Storage API için tarayıcı destek durumu:

Tarayıcı	Sürümü
Internet Explorer	8+
Firefox	2+
Opera	10.50+
Safari (WebKit)	4+